Supplementary Table 1. Adapter and primer sequences. a) Illumina Nextera primers used for all libraries. b) Novel primers for plant and algal ITS1 regions. c) Adapters, second round of PCR. d) Barcode index sequences.

a)

Adapter	Sequence
PrefixNX/F	5'-TCGTCGGCAGCGTC AGATGTGTATAAGAGACAG-3'
PrefixNX/R	5'-GTCTCGTGGGCTCGG AGATGTGTATAAGAGACAG-3'

b)

Primer	Direction	Target length	Sequence
ITS1-F	forward	--	5'-GTCGTAACAAGGTTTCCGTAGGT-3'
ITS1-R3	reverse	240-500 bp	5'-GATATCCGTTGCCG[AG]GAGTC-3'

c)

Adapter	Sequence
NX/F	5'-AATGATACGGCGACCACCGAGATCTACAC[i5]TCGTCGGCAGCGTC-3'
NX/R	5'-CAAGCAGAAGACGGCATACGAGAT[i7]GTCTCGTGGGCTCGG-3'

d)

[i5]		[i7]	
S502	CTCTCTAT	N701	TCGCCTTA
S503	TATCCTCT	N702	CTAGTACG
S504	AGAGTAGA	N703	TTCTGCCT
S505	GTAAGGAG	N704	GCTCAGGA
S506	ACTGCATA	N705	AGGAGTCC
S507	AAGGAGTA	N706	CATGCCTA
S508	CTAAGCCT	N707	GTAGAGAG
S517	GCGTAAGA	N708	CCTCTCTG
		N709	AGCGTAGC
		N710	CAGCCTCG
		N711	TGCCTCTT
		N712	TCCTCTAC

Supplementary Table 2. Species and sequences used in novel ITS1 primer design.

Genus	Species	Region	GenBank ID
Andromeda	polifolia	ITS1	AF358872
Bidens	beckii	ITS1	U67096
Bidens	frondosa	ITS1	U67094
Brasenia	schreberi	18S	AF206874
Brasenia	schreberi	ITS1	EF526394
Calla	palustris	18S	AF168829
Callitriche	heterophylla	18S	AF107582
Caltha	palustris	26S	U52632
Caltha	palustris	ITS1	KF233852
Campanula	ramulosa	18S	U42510
Cardamine	pensylvatica	ITS1	DQ268469
Carex	aquatilis	5.8S	FJ904611
Carex	bebbii	ITS1	AY779071
Carex	brunnescens	ITS1	AY757405
Carex	cannescens	ITS1	AY757406
Carex	chordorrhiza	ITS1	AY757409
Carex	comosa	ITS1	AY757575
Carex	crawfordii	ITS1	AY779089
Carex	crinita	ITS1	AY757589
Carex	diandra	ITS1	AY699626
Carex	echinata	ITS1	EU352224
Carex	flava	ITS1	AY278310
Carex	grayi	ITS1	AY757580
Carex	interior	ITS1	AY757419
Carex	lasiocarpa	ITS1	AY278297
Carex	leptalea	ITS1	AF285060
Carex	limosa	ITS1	AY278298
Carex	lupulina	ITS1	AF284963
Carex	magellanica	ITS1	AY278292
Carex	oligosperma	ITS1	AY757578
Carex	pauciflora	ITS1	GU176161
Carex	pedunculata	ITS1	AF284969
Carex	projecta	ITS1	AY757423
Carex	pseudocyperus	ITS1	EU352234
Carex	retrorsa	ITS1	AY757577
Carex	rostrata	ITS1	JN314569
Carex	scoparia	ITS1	AY779156
Carex	sterilis	ITS1	DQ115286
Carex	stipata	ITS1	AY757426
Carex	stricta	ITS1	AY770487
Carex	tribuloides	ITS1	AY757428
Carex	trisperma	ITS1	AY757429
Carex	tuckermanii	ITS1	AY757573
Carex	vesicaria	ITS1	AY278289
Carex	vesicaria	ITS1	JN314544
Ceratophyllum	demersum	18S	D85300
Ceratophyllum	demersum	28S	X83843
Ceratophyllum	demersum	ITS1	KM582605
Cicuta	bulbifera	ITS1	AY524708
Cicuta	bulbifera	ITS1	AY524714
Cicuta	maculata	18S	KT179692
Decodon	verticillatus	ITS1	AY905421
Dulichium	arundinaceum	ITS1	DQ998949
Elatine	triandra	ITS1	AY674594
Eleocharis	acicularis	ITS1	GU977052
Eleocharis	elliptica	ITS1	GU110805
Eleocharis	obtusa	ITS1	AF190595
Eleocharis	ovata	ITS1	GU977091
Eleocharis	palustris	ITS1	GU977092
Eliocaulon	cinereum	ITS1	EU924280
Elodea	canadensis	18S	AF168841
Elodea	canadensis	ITS1	HQ456421
Erica	x stuartii	ITS1	KP737631
Eriocaulon	septangulare	ITS1	AY952402
Glyceria	striata	ITS1	EU594618
Hydrophyllum	capitatum	18S	HQ384691
Hydrophyllum	tenuipes	ITS1	AF091170
Juncus	articulatus	ITS1	AY727819
Juncus	brevicaudatus	ITS1	AY727823
Juncus	canadensis	ITS1	AY727814
Juncus	effusus	18S	AF206944
Juncus	effusus	ITS1	FJ980286
Juncus	filiformis	ITS1	AY727790
Juncus	pelocarpus	ITS1	AY727811
Lemna	minor	ITS1	KJ400888
Littorella	uniflora	ITS1	AJ548962
Ludwigia	palustris	18S	AM235545
Ludwigia	palustris	ITS1	FN263234
Lycopus	uniflorus	ITS1	DQ667302
Lysimachia	thrysiflora	ITS1	AF547725
Lythrum	salicaria	18S	AF206955
Lythrum	salicaria	ITS1	AY035750
Myosotis	scorpioides	ITS1	EU594655
Myriophyllum	exalbescens	18S	U42551
Myriophyllum	farwellii	ITS1	EF178731
Myriophyllum	heterophyllum	ITS1	EF526366
Myriophyllum	sibirisum	ITS1	FJ426357
Myriophyllum	spicatum	ITS1	EF526362
Myriophyllum	tenellum	ITS1	EF526410
Najas	flexilis	ITS1	HM240422
Nitellopsis	obtusa	18S	AY823710
Nuphar	lutea	ITS1	AY620427
Nymphaea	odorata	18S	AF206973
Nymphaea	odorata	ITS1	EF526395
Phacelia	ranunculacea	ITS1	DQ005986
Pontederia	cordata	18S	AF206998
Potamogeton	amplifolius	ITS1	EF526388
Potamogeton	berchtoldii	18S	DQ007410
Potamogeton	diversifolius	ITS1	FJ151205
Potamogeton	epihydrus	ITS1	FJ151207
Potamogeton	foliosus	ITS1	KF270907
Potamogeton	gramineus	ITS1	AB491756
Potamogeton	natans	ITS1	JX012090
Potamogeton	oakesianus	ITS1	FJ151213
Potamogeton	obtusifolius	ITS1	AB744004
Potamogeton	perfoliatus	18S	AY952389
Potamogeton	praelongus	ITS1	AB744005
Potamogeton	pusillus	ITS1	KF270914
Potamogeton	richardsonii	ITS1	EU596955
Potamogeton	robbinsii	ITS1	EF526390
Potamogeton	spirillus	ITS1	EF526373
Potamogeton	vaseyi	ITS1	GQ247437
Potamogeton	zosteriformis	ITS1	KF270921
Rorippa	palustris	ITS1	KC133366
Sagittaria	latifolia	ITS1	HQ456403
Sagittaria	latifolia	ITS1	JF780975
Sagittaria	trifolia	18S	D29781
Schoenoplectus	acutus	ITS1	KC677953
Schoenoplectus	americanus	ITS1	KC677959
Schoenoplectus	tabernaemontani	ITS1	DQ385592
Sium	suave	ITS1	AY548213
Spirodela	polyrhiza	18S	KJ400913
Symphyotrichum	boreale	ITS1	EU781456
Syngonanthus	cipoensis	ITS1	GQ861448
Triadenum	fraseri	ITS1	HE653668
Triadenum	walteri	18S	FJ669669
Typha	latifolia	18S	AF168880
Typha	latifolia	ITS1	KF265390
Utricularia	bifida	ITS1	KF016005
Utricularia	sp.Albach	18S	AJ236044
Vallisneria	americana	18S	AF069201
Vallisneria	americana	ITS1	EF526407

Supplementary Table 3. Potential false-positive plant taxa. These six records represent the only vascular genera in the eDNA dataset not previously recorded at the UNDERC study sites.

Family	Scientific Name	Common Name	Total Reads	Present in	
				Samples	Lakes
a) Possibly missing in error from UNDERC inventory					
Moraceae	Morus	mulberries	444	2	2
b) Grasses, possibly recently invasive or misidentified in eDNA results					
Poaceae	Briza brasiliensis	Brazilian quaking-grass	352	1	1
Poaceae	Festuca luciarum	fescue	241	1	1
c) Highly cosmopolitan or agricultural species					
Asteraceae	Ambrosia artemisiifolia	annual ragweed	897	1	1
Fabaceae	Cyamopsis tetragonoloba	guar	1684	1	1
Fabaceae	Glycine max	soybean	399	14	5

Supplementary Table 4. Macrophyte presences by lake. Wetland statuses follow USDA/USACoE conventions, plus an additional category "AQU" for obligate wetland species that are fully aquatic, i.e., floating and submerged macrophytes.

Family	Taxon Name	Common Name	Wetland Status	Lake Presence (# Samples)					
				Bay	Hmg	Lng	IP	Mor	Ras
Basal streptophytes and conifers									
Ceratophyllaceae	*Ceratophyllum sp.*	coontails	AQU				2		
Ceratophyllaceae	*Ceratophyllum platyacanthum*	coontail	AQU				1		
Cabombaceae	*Brasenia schreberi*	Watershield	AQU	1	3				
Nymphaeaceae	*Nuphar variegata*	Yellow Pond-Lily	AQU	4	12		9	10	2
Nymphaeaceae	*Nymphaea odorata*	American White Water-Lily	AQU		1	1		8	
Pinaceae	*Pinus armandii*	Chinese White Pine	FACU			2			1
Monocots									
Amaryllidaceae	*Allium sativum*	garlic	FACU	1				1	
Alismataceae	*Sagittaria latifolia*	Duck-Potato	AQU				2		
Hydrocharitaceae	*Elodea bifoliata*	Two-Leaf Waterweed	AQU				1		3
Hydrocharitaceae	*Elodea nuttallii*	Western Waterweed	AQU				1		1
Hydrocharitaceae	*Najas sp.*	naiads	AQU		1		2	3	
Hydrocharitaceae	*Najas flexilis*	Wavy Waternymph	AQU				2	1	
Potamogetonaceae	*Potamogeton sp.*	pondweeds	AQU		11	1	5	7	6
Potamogetonaceae	*Potamogeton lucens*	Illinois Pondweed	AQU		1		7	9	
Potamogetonaceae	*Potamogeton oakesianus*	Oakes' Pondweed	AQU				5	3	1
Potamogetonaceae	*Potamogeton richardsonii*	Red-Head Pondweed	AQU				4		
Potamogetonaceae	*Potamogeton spirillus*	Spiral Pondweed	AQU				2	1	
Potamogetonaceae	*Potamogeton zosteriformis*	Flat-Stem Pondweed	AQU		2		3	3	1
Cyperaceae	*Carex echinata*	Star Sedge	OBL		2		1		
Cyperaceae	*Carex trisperma*	Three-Seed Sedge	OBL		1				
Cyperaceae	*Dulichium arundinaceum*	Three-Way Sedge	OBL			1			
Poaceae	*Agrostis sp.*	bentgrasses	FACU	1		1			3
Poaceae	*Calamagrostis angustifolia*	Narrow-Leaf Bent-Grass	OBL	2	1	2	3	1	1
Poaceae	*Briza brasiliensis*	Brazilian Quaking-Grass	FACU				1		
Poaceae	*Cynodon dactylon*	Bermuda Grass	FACU					1	
Poaceae	*Festuca luciarum*	fescue	FACU				1		
Poaceae	*Glyceria canadensis*	Rattlesnake Manna Grass	OBL						1
Poaceae	*Lolium perenne*	Perennial Rye Grass	FACU				1		
Poaceae	*Phleum pratense*	Common Timothy	FACU	1	1		2		1
Poaceae	*Poa annua*	Annual Blue Grass	FACU			1			
Poaceae	*Poa compressa*	Flat-Stem Blue Grass	FACU				1		
Poaceae	*Poa infirma*	Annual Blue Grass	FACU			1			
Poaceae	*Poa pratensis*	Kentucky Blue Grass	FAC		2		2		1
Poaceae	*Poa trivialis*	Rough Bluegrass	FACU		1				

Supplementary Table 4 cont.

Family	Taxon Name	Common Name	Wetland Status	Lake Presence (# Samples)					
				Bay	Hmg	Lng	IP	Mor	Ras
Dicots									
Apiaceae	Apiaceae sp.	parsley family	FACU						1
Apiaceae	*Cicuta bulbifera*	Bulblet-Bearing Water-Hemlock	AQU				2		
Aquifoliaceae	*Ilex sp.*	hollies	FACU			1			
Aquifoliaceae	*Ilex serrata*	Japanese Winterberry	FACU		1	2			
Asteraceae	*Helianthus sp.*	sunflowers	FAC				6		
Asteraceae	*Taraxacum sp.*	dandelions	FAC	1	1		1		
Asteraceae	*Ambrosia artemisiifolia*	Annual Ragweed	FACU		1				
Caryophyllaceae	*Stellaria graminea*	Grass-Leaf Starwort	UPL			1			
Balsaminaceae	*Impatiens capensis*	Spotted Touch-Me-Not	FACW				2		
Ericaceae	*Chamaedaphne calyculata*	Leatherleaf	OBL		1	1		1	
Fabaceae	*Cyamopsis tetragonoloba*	guar	FACU				1		
Fabaceae	*Glycine max*	soybean	FACU	2	1	4	2	2	
Fabaceae	*Medicago lupulina*	Black Medick	FACU				1		
Fabaceae	*Vicia faba*	fava bean	FACU					1	
Betulaceae	*Alnus sp.*	alders	FAC				2		
Betulaceae	*Betula sp.*	birches	FAC			1			
Betulaceae	*Betula pumila*	Bog Birch	OBL			1	4	2	1
Juglandaceae	*Carya illinoinensis*	Pecan	FACW				1		
Myricaceae	*Myrica gale*	Sweetgale	OBL	2	6	2			
Lentibulariaceae	*Utricularia sp.*	bladderworts	FAC					1	3
Plantaginaceae	*Plantago sp.*	plantains	FAC		1				
Hypericaceae	Hypericaceae sp.	St-Johns-Wort Family	FACU				1		
Ranunculaceae	*Ranunculus sp.*	buttercups	FAC						1
Ranunculaceae	*Ranunculus sceleratus*	Cursed Buttercup	OBL						1
Moraceae	*Morus sp.*	mulberries	FAC		1				
Rosaceae	*Rubus allegheniensis*	Allegheny Blackberry	FACU				1		
Urticaceae	*Urtica angustifolia*	Stinging Nettle	FACW			2			
Urticaceae	*Urtica gracilis*	Stinging Nettle	FACW	4	2	2	1	1	1
Haloragaceae	*Myriophyllum farwellii*	Farwell's Water-Milfoil	AQU	1					1

Supplementary Table 5. Phytoplankton presences by lake. Bold/highlighted items are the top ten phytoplankton taxa overall by read abundance.

Taxon Name	Higher Classifications					Lake Presence (# Samples out of 12)						Total Reads	NCBI Taxon Accession
						Bay	Hmg	Lng	IP	Mor	Ras		
Viridiplantae													
Chlorophyceae													
Aphanochaete	Chlorophyta	Chlorophyceae	Chaetophorales	Aphanochaetaceae			1	3	2	3	32	104530	
Chaetophorales	Chlorophyta	Chlorophyceae	Chaetophorales		1	1	2	2			13	31299	
Chlamydomonadaceae	Chlorophyta	Chlorophyceae	Chlamydomonadales	Chlamydomonadaceae	7	12	12	9	5	12	4487	3051	
Chlamydomonadaceae sp01	Chlorophyta	Chlorophyceae	Chlamydomonadales	Chlamydomonadaceae	1	3	8	6	1	4	384	225504	
Chlamydomonadaceae sp03	Chlorophyta	Chlorophyceae	Chlamydomonadales	Chlamydomonadaceae	12		12	1		10	1551	225505	
Chlamydomonadaceae sp04	Chlorophyta	Chlorophyceae	Chlamydomonadales	Chlamydomonadaceae			4	1	2	1	20	225416	
Chlamydomonadaceae sp05	Chlorophyta	Chlorophyceae	Chlamydomonadales	Chlamydomonadaceae	3	2	10	3		4	65	225522	
Chlamydomonadaceae sp06	Chlorophyta	Chlorophyceae	Chlamydomonadales	Chlamydomonadaceae			5	4	2	1	41	225409	
Chlamydomonas	Chlorophyta	Chlorophyceae	Chlamydomonadales	Chlamydomonadaceae	3	12	12	11	7	10	1660	3052	
Chlamydomonas leiostraca	Chlorophyta	Chlorophyceae	Chlamydomonadales	Chlamydomonadaceae	1	1	3		1	3	28	1034604	
Chlamydomonas monadina	Chlorophyta	Chlorophyceae	Chlamydomonadales	Chlamydomonadaceae	4	1	7	1	2	3	57	47904	
Chlamydomonas nasuta	Chlorophyta	Chlorophyceae	Chlamydomonadales	Chlamydomonadaceae	3		2			1	20	1157966	
Chlamydomonas sp02	Chlorophyta	Chlorophyceae	Chlamydomonadales	Chlamydomonadaceae	1	5	5	4	2	1	65	1810845	
Chloromonas	Chlorophyta	Chlorophyceae	Chlamydomonadales	Chlamydomonadaceae	1	12	2	2		3	1399	1219481	
Lobochlamys segnis	Chlorophyta	Chlorophyceae	Chlamydomonadales	Chlamydomonadaceae	1	1	1	1	2		10	52035	
Rhysamphichloris curta	Chlorophyta	Chlorophyceae	Chlamydomonadales	Chlamydomonadaceae	2		3			1	12	1815954	
Chlorococcum diplobionticum	Chlorophyta	Chlorophyceae	Chlamydomonadales	Chlorococcaceae		1	2	2	1	3	19	52026	
Hafniomonas reticulata	Chlorophyta	Chlorophyceae	Chlamydomonadales	Dunaliellaceae				3		1	25	51322	
Chlamydomonadaceae sp02	Chlorophyta	Chlorophyceae	Chlamydomonadales	Ploeotila						2	168	687530	
Pandorina morum	Chlorophyta	Chlorophyceae	Chlamydomonadales	Volvocaceae				8	1		24	33099	
Oedogonium	Chlorophyta	Chlorophyceae	Oedogoniales		1	6	1		3	5	149	55993	
Oedogonium howardii	Chlorophyta	Chlorophyceae	Oedogoniales			1			3	3	24	468365	
Oedogonium sp02	Chlorophyta	Chlorophyceae	Oedogoniales		2		2		3	5	43	1429381	
Oedogonium stellatum	Chlorophyta	Chlorophyceae	Oedogoniales			5	1		1	4	41	337029	
Oedogonium subplagiostomum	Chlorophyta	Chlorophyceae	Oedogoniales			1				1	19	337946	
Oedogoniales	Chlorophyta	Chlorophyceae	Oedogoniales		2	11	2		3	5	849	35490	
Hydrodictyaceae	Chlorophyta	Chlorophyceae	Sphaeropleales	Hydrodictyaceae			6	10	11	7	1085	3103	
Neochloris sp01	Chlorophyta	Chlorophyceae	Sphaeropleales	Neochloridaceae	5		10	8	3	6	182	1475470	
Polyedriopsis spinulosa	Chlorophyta	Chlorophyceae	Sphaeropleales	Neochloridaceae				4		2	10	162320	
Follicularia texensis	Chlorophyta	Chlorophyceae	Sphaeropleales	Radiococcaceae	9			10		2	132	1082342	
Radiococcus polycoccus	Chlorophyta	Chlorophyceae	Sphaeropleales	Radiococcaceae	12	2	10	10	2	12	**7243**	170400	
Chlorophyceae *cont.*													
Desmodesmus bicellularis	Chlorophyta	Chlorophyceae	Sphaeropleales	Scenedesmaceae	5		12	11	7	9	210	91203	
Verrucodesmus parvus	Chlorophyta	Chlorophyceae	Sphaeropleales	Scenedesmaceae					3		129	1034620	
Selenastraceae	Chlorophyta	Chlorophyceae	Sphaeropleales	Selenastraceae	7		2	7	11	6	226	35466	
Sphaeropleaceae	Chlorophyta	Chlorophyceae	Sphaeropleales	Sphaeropleaceae	12		12	11		11	**9928**	52675	
uncultured Chlorophyceae	Chlorophyta	Chlorophyceae			3			2			13	485336	
Chlorophyta sp01	Chlorophyta					4					18	429243	
Chlorophyta sp02	Chlorophyta				5		11	7	2	5	162	1429408	
other chlorophytes													
Scherffelia dubia	Chlorophyta	Chlorodendrophyceae	Chlorodendrales	Chlorodendraceae				3	1		90	3190	
Monomastix opisthostigma	Chlorophyta	Mamiellophyceae	Monomastigales	Monomastigaceae	1		6	1	1	1	14	687946	
Botryococcus braunii	Chlorophyta	Trebouxiophyceae		Botryococcaceae	2		3	1	2	1	30	38881	
Coenochloris sp01	Chlorophyta	Trebouxiophyceae	Chlorellales	Oocystaceae	4		1	2		2	40	798156	
Oocystis sp01	Chlorophyta	Trebouxiophyceae	Chlorellales	Oocystaceae	3		2	10	1	1	64	1912763	
Willea vilhelmii	Chlorophyta	Trebouxiophyceae	Chlorellales		12		3			7	1224	1911616	
Chlorellales	Chlorophyta	Trebouxiophyceae	Chlorellales					3	6		18	35460	
Choricystis	Chlorophyta	Trebouxiophyceae			6	10	12	8	10	3	438	41299	
uncultured prasinophyte	Chlorophyta	Trebouxiophyceae	Chlorellales		11		12		2	11	621	157820	
Scotinosphaera sp01	Chlorophyta	Ulvophyceae	Scotinosphaerales	Scotinosphaeraceae	1		5	1	1	3	41	1199106	
Ulotrichales	Chlorophyta	Ulvophyceae	Ulotrichales		2			1	1	2	12	31306	
charophytes													
Chara braunii	Streptophyta	Charophyceae	Charales	Characeae					6		42	69332	
Coleochaete nitellarum	Streptophyta	Coleochaetophyceae	Coleochaetales	Coleochaetaceae	1	1	1	4	1	2	21	78178	
Closterium incurvum	Streptophyta	Zygnemophyceae	Desmidiales	Closteriaceae	1	1		1	3	4	56	157578	
Closterium pusillum	Streptophyta	Zygnemophyceae	Desmidiales	Closteriaceae		2			1	2	14	157573	
Closterium spinosporum	Streptophyta	Zygnemophyceae	Desmidiales	Closteriaceae				2	2		24	157566	
Closterium venus	Streptophyta	Zygnemophyceae	Desmidiales	Closteriaceae				2	2	2	10	157582	
Cosmarium	Streptophyta	Zygnemophyceae	Desmidiales	Desmidiaceae	1		2		1	3	16	33100	
Cosmarium regnellii	Streptophyta	Zygnemophyceae	Desmidiales	Desmidiaceae	2	3	1		3	4	33	485958	
Desmidium swartzii	Streptophyta	Zygnemophyceae	Desmidiales	Desmidiaceae	1		1			3	123	184503	
Euastrum	Streptophyta	Zygnemophyceae	Desmidiales	Desmidiaceae	6		12			5	473	130992	
Hyalotheca dissiliens	Streptophyta	Zygnemophyceae	Desmidiales	Desmidiaceae	1	1	3			4	28	130987	
Spondylosium secedens	Streptophyta	Zygnemophyceae	Desmidiales	Desmidiaceae	8		9			6	87	184519	

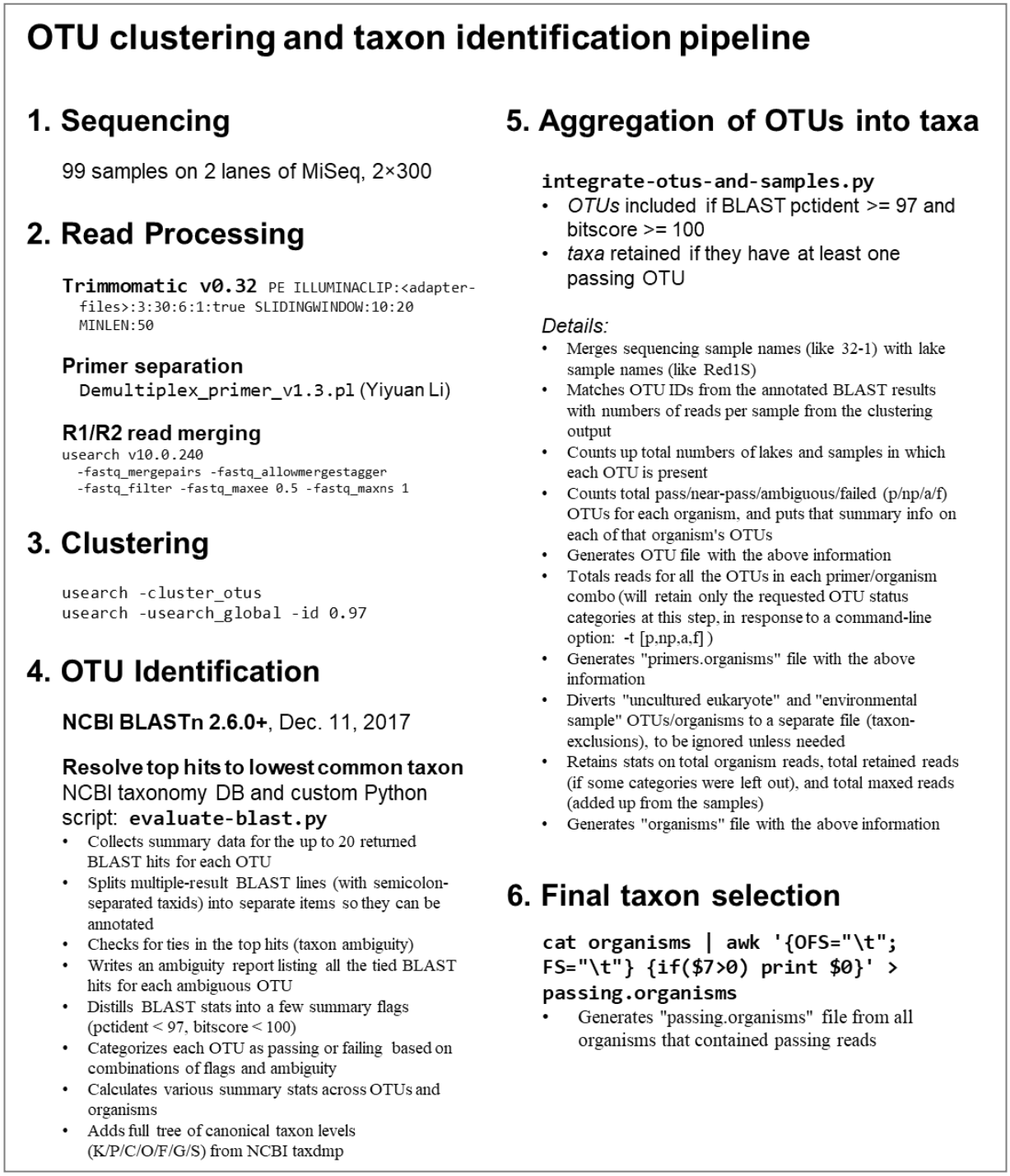
Supplementary Table 5 cont.

Taxon Name	Higher Classifications				Lake Presence (# Samples out of 12)						Total Reads	NCBI Taxon Accession
					Bay	Hmg	Lng	IP	Mor	Ras		
Viridiplantae *cont.*												
charophytes *cont.*												
Staurastrum	Streptophyta	Zygnemophyceae	Desmidiales	Desmidiaceae	12	6	12	6	4	12	2085	31317
Staurodesmus extensus	Streptophyta	Zygnemophyceae	Desmidiales	Desmidiaceae	12	1	11		3	12	729	485991
Gonatozygon monotaenium	Streptophyta	Zygnemophyceae	Desmidiales	Gonatozygaceae			2			3	14	52235
Mougeotia	Streptophyta	Zygnemophyceae	Zygnematales	Zygnemataceae						2	35	13157
Eukaryota												
diatoms												
Achnanthidium	Bacillariophyta	Bacillariophyceae	Cocconeidales	Achnanthidiaceae				1	3		11	265567
Cymbella cistula	Bacillariophyta	Bacillariophyceae	Cymbellales	Cymbellaceae				1	2		42	930063
Gomphonema carolinense	Bacillariophyta	Bacillariophyceae	Cymbellales	Gomphonemataceae				1	3		21	1574446
Gomphonema clavatum	Bacillariophyta	Bacillariophyceae	Cymbellales	Gomphonemataceae					2		32	1223577
Gomphonema sp01	Bacillariophyta	Bacillariophyceae	Cymbellales	Gomphonemataceae				1	2		37	1574466
Eunotia sp01	Bacillariophyta	Bacillariophyceae	Eunotiales	Eunotiaceae	1				4		25	1629841
Navicula	Bacillariophyta	Bacillariophyceae	Naviculales	Naviculaceae				2	3		14	50949
Aulacoseira	Bacillariophyta	Coscinodiscophyceae		Aulacoseiraceae				2	3		26	49233
Melosira ambiqua	Bacillariophyta	Coscinodiscophyceae	Melosirales	Melosiraceae				12	1		**24524**	70449
Urosolenia eriensis	Bacillariophyta	Coscinodiscophyceae	Rhizosoleniales	Rhizosoleniaceae					2		17	1003149
Fragilariaceae	Bacillariophyta	Fragilariophyceae	Fragilariales	Fragilariaceae				2	6		27	33856
Synedra berolinensis	Bacillariophyta	Fragilariophyceae	Fragilariales	Fragilariaceae				1	2		17	436006
Licmophora reichardtii	Bacillariophyta	Fragilariophyceae		Licmophoraceae	1	1	3			1	14	427314
Fragilariophyceae	Bacillariophyta	Fragilariophyceae			11	1	4	1	6	3	130	33853
golden algae												
Chromulina sp01		Chrysophyceae	Chromulinales	Chromulinaceae	5		8			5	39	420570
Ochromonas		Chrysophyceae	Chromulinales	Chromulinaceae	1	1	3	3	8	3	258	2985
Dinobryon		Chrysophyceae	Chromulinales	Dinobryaceae			9	7	1	8	2222	98059
Dinobryon pediforme		Chrysophyceae	Chromulinales	Dinobryaceae	11	3	2	8	2	6	282	455251
Dinobryon sp01		Chrysophyceae	Chromulinales	Dinobryaceae	7	7	9	8	1	9	857	1955563
Epipyxis		Chrysophyceae	Chromulinales	Dinobryaceae	8		4	5		8	120	64705
Kephyrion sp01		Chrysophyceae	Chromulinales	Dinobryaceae	9		9	4	4	5	1075	1074837
Paraphysomonas sp01		Chrysophyceae	Chromulinales	Paraphysomonadaceae				1	9		1474	36769
Paraphysomonas sp02		Chrysophyceae	Chromulinales	Paraphysomonadaceae	10		8	9	11	4	2203	1955561
Chrysophyceae sp03		Chrysophyceae		Chrysophyceae	12	7	12	10	8	12	**95582**	1955566
Chrysophyceae sp01		Chrysophyceae		Chrysophyceae	2	6	1		1	5	48	715244
Chrysosaccus sp01		Chrysophyceae	Chrysosphaerales	Chrysosaccaceae		1			2	5	22	420574
Chrysophyceae sp05		Chrysophyceae			12	10	11	11	8	12	6241	183512
Chrysophyceae sp04		Chrysophyceae			12		6	6	5	3	330	2825
Cryptomonas	Cryptophyta		Cryptomonadales	Cryptomonadaceae			1	5	9	1	161	3030
Cryptomonas ovata	Cryptophyta		Cryptomonadales	Cryptomonadaceae				2	2	3	75	70450
Cryptomonas phaseolus	Cryptophyta		Cryptomonadales	Cryptomonadaceae	1		1	2	3	1	146	400110
Cryptomonas pyrenoidifera	Cryptophyta		Cryptomonadales	Cryptomonadaceae	11	4	10	6	7	9	567	233184
Cryptomonas tetrapyrenoidosa	Cryptophyta		Cryptomonadales	Cryptomonadaceae	12	8	11	2	7	10	707	233185
uncultured Cryptophyta	Cryptophyta				1		4	1	5		26	243282
dinoflagellates												
Ceratium hirundinella		Dinophyceae	Gonyaulacales	Ceratiaceae	1		12	11	12	5	**51510**	261830
Gymnodinium sp01		Dinophyceae	Gymnodiniales	Gymnodiniaceae	8		10	8	2	10	3323	304864
Gymnodiniaceae sp01		Dinophyceae	Gymnodiniales	Gymnodiniaceae			2	2	5	3	127	926437
Woloszynskia leopoliensis		Dinophyceae	Lophodiniales	Lophodiniaceae			3			3	20	261834
Glenodiniopsis steinii		Dinophyceae	Peridiniales	Glenodiniaceae				1	1	1	25	160615
Peridinium		Dinophyceae	Peridiniales	Peridiniaceae	7	3	12	2	6	7	635	2867
Peridinium gatunense		Dinophyceae	Peridiniales	Peridiniaceae		1		2	3		13	185948
Peridinium limbatum		Dinophyceae	Peridiniales	Peridiniaceae		10	7	7	9	7	4132	407207
Peridinium wierzejskii		Dinophyceae	Peridiniales	Peridiniaceae	12	12	12	11	12	12	**1E+05**	261832
Peridiniopsis polonicum		Dinophyceae	Peridiniales	Peridiniopsidaceae	12	2	4	5	11	9	1644	439559
Scrippsiella sp01		Dinophyceae	Peridiniales	Thoracosphaeraceae			5	5	10		1104	427755
Thoracosphaera heimii		Dinophyceae	Peridiniales	Thoracosphaeraceae	11	6	8	7	12	9	**14054**	2923
Spiniferodinium galeiforme		Dinophyceae	Phytodiniales	Phytodiniaceae	10		5			3	66	673113
Phytodiniaceae		Dinophyceae	Phytodiniales	Phytodiniaceae	3						11	425788
Piscinoodinium sp01		Dinophyceae	Suessiales	Suessiaceae	7		3			3	150	419288
Suessiaceae sp01		Dinophyceae	Suessiales	Suessiaceae	7		1				54	1691401
uncultured dinoflagellate		Dinophyceae			12	11	12	11	9	12	**7850**	187300
Dinophyceae		Dinophyceae			10	1	7	2	9	5	1172	2864

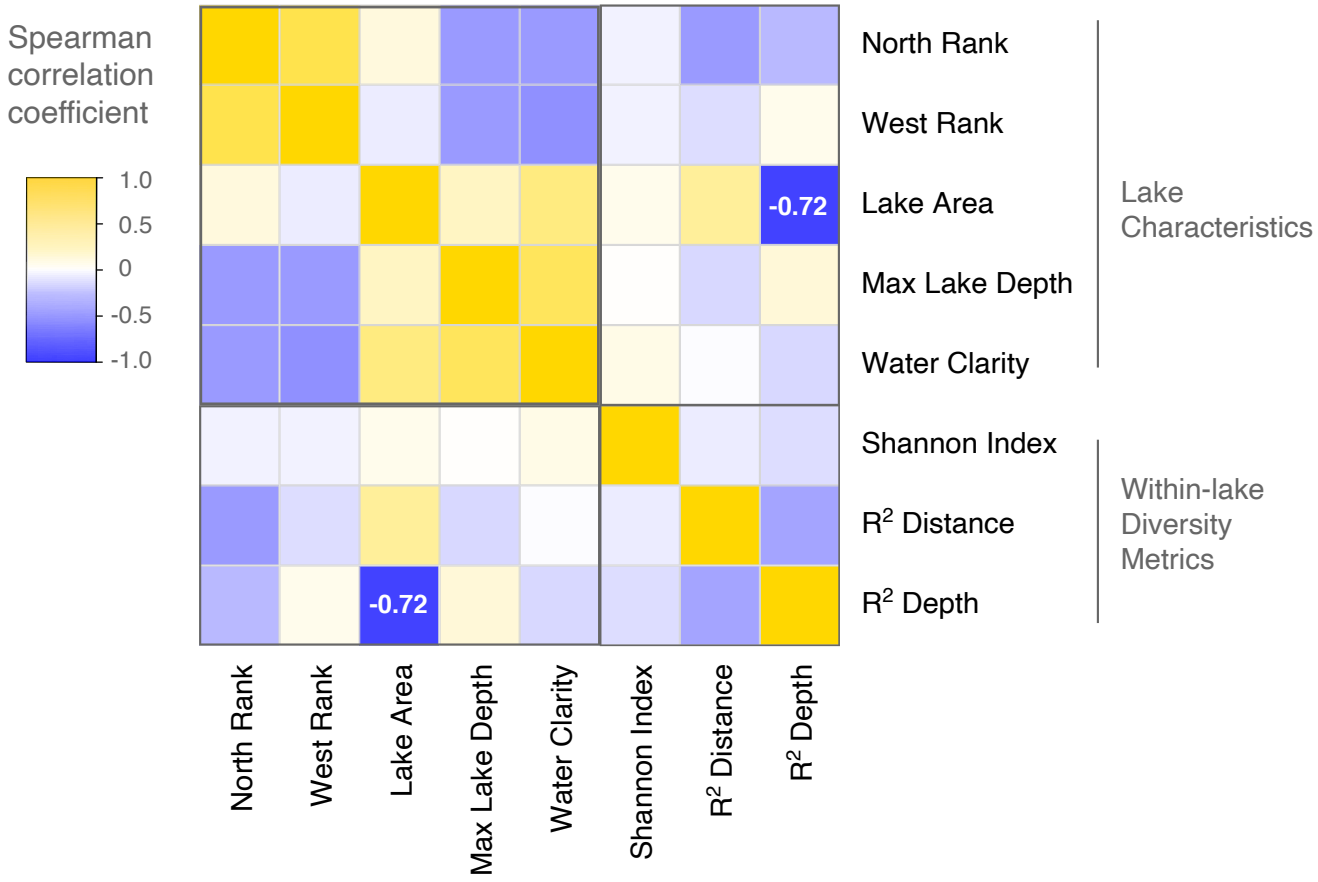
Supplementary Table 5 cont.

Taxon Name	Higher Classifications				Lake Presence (# Samples out of 12)						Total Reads	NCBI Taxon Accession
					Bay	Hmg	Lng	IP	Mor	Ras		
Eukaryota *cont.*												
other eukaryotes												
Gonyostomum semen	Raphidophyceae	Chattonellales	Vacuolariaceae		9	12	12	11	12	10	**76996**	375454
Mallomonas	Synurophyceae	Synurales	Mallomonadaceae		9	8	10	6	8	8	2270	2988
Mallomonas sp01	Synurophyceae	Synurales	Mallomonadaceae				8	3			36	2003107
Synura	Synurophyceae	Synurales	Mallomonadaceae		2		8	5			213	2991
Synura mammillosa	Synurophyceae	Synurales	Mallomonadaceae		2	12	9	2		8	**55712**	52554
Synura spinosa	Synurophyceae	Synurales	Mallomonadaceae				1	4			53	2992
Synura uvella	Synurophyceae	Synurales	Mallomonadaceae		1		10				43	52557
Chrysosphaerella brevispina	Synurophyceae	Synurales	Synuraceae						9		78	1261421
uncultured Synurophyceae	Synurophyceae				9	5	8	7	10	9	1100	1138870
Pavlova noctivaga		Pavlovales	Pavlovaceae		2		2			1	12	349099
Chrysochromulina parva		Prymnesiales	Chrysochromulinaceae					10	9		966	127563
Chrysochromulina sp01		Prymnesiales	Chrysochromulinaceae					10	8		563	1460289
uncultured haptophyte					1		4	1			13	171259

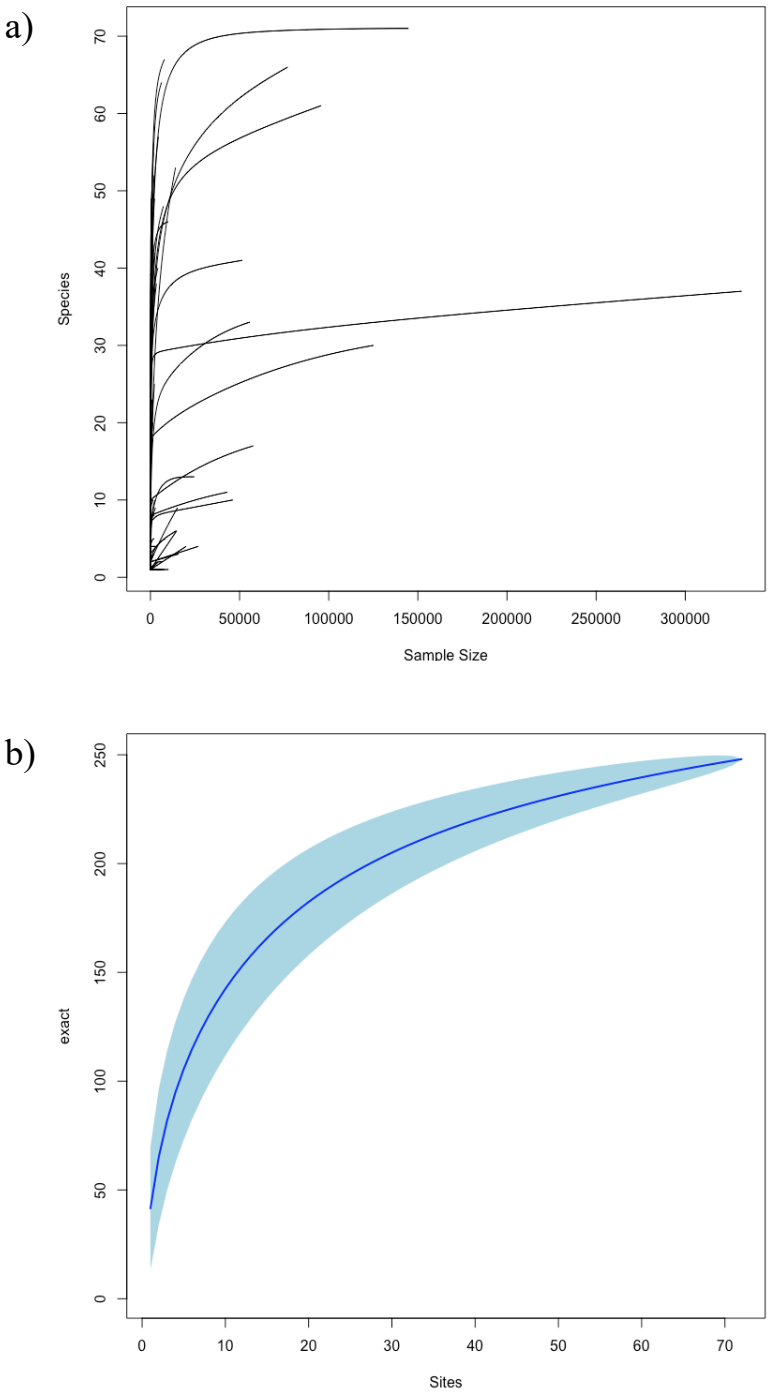
Supplementary Figure 1. Details of OTU clustering and identification pipeline. Software versions and descriptions of custom scripts for clustering and BLAST pipeline, including ambiguity resolution



Supplementary Figure 2. Spearman correlations with diversity indices among lake covariates. The only factors that display substantial correlation across lake characteristics and biodiversity metrics are the area of the lake and the strength of its depth effect. However, the effect is not statistically significant (P=0.195), nor are any other correlations across lake characteristics and diversity metrics.



Supplementary Figure 3. Rarefaction and species accumulation curves.
a) Rarefaction curves by read number and per lake. b) Species accumulation curve by sample over entire study.



```
#!/usr/bin/perl -w

# Demultiplex_primer_v1.3.pl

use strict;

die "Usage: perl $0 [1.fq] [2.fq] [primer_set.fas] [output prefix] [report]\n" unless (@ARGV == 5);

my (%hash, $line1, $line2);
open (PRI, $ARGV[2]) or die "$ARGV[2] $!\n";
open (DE, ">$ARGV[4]") or die "$ARGV[4] $!\n";
open (LG, ">$ARGV[3].log") or die "$ARGV[3].log $!\n";
$/ = '>';
<PRI>;
while(<PRI>){
    chomp;
    my @line = split /\n+/;

# To remove weird linebreaks
    $line[0] =~ s/\s//g;
    $line[1] =~ s/\s//g;

    my @id = split /\_/, $line[0];
    $hash{$id[0]}{$id[1]} = $line[1];

}
close PRI;
$/ = "\n";

# print "Primers\t1.fastq\t2.fastq\tpaired_reads\n";
print DE "Primers\t1.fastq\t2.fastq\tpaired_reads\n";

foreach my $k (sort keys %hash){
    if ($ARGV[0] =~ /gz$/){
        open (FQ1, "gzip -dc $ARGV[0] |") or die "$ARGV[0] $!\n";
    }else{
        open (FQ1, $ARGV[0]) or die "$ARGV[0] $!\n";
    }

    if ($ARGV[1] =~ /gz$/){
        open (FQ2, "gzip -dc $ARGV[1] |") or die "$ARGV[1] $!\n";
    }else{
        open (FQ2, $ARGV[1]) or die "$ARGV[1] $!\n";
    }

# print "$k\t";
print DE "$k\t";

    open (OT1, ">$ARGV[3]_$k.F.fq") or die "$ARGV[3]_$k.F.fq $!\n";
    open (OT2, ">$ARGV[3]_$k.R.fq") or die "$ARGV[3]_$k.R.fq $!\n";

my ($fq1_F, $fq1_R, $fq2_F, $fq2_R);
my $fq1 = 0;
my $fq2 = 0;
my $pd = 0;

    while(my $forward1 = <FQ1>){
        my $forward2 = <FQ1>;
        my $forward3 = <FQ1>;
        my $forward4 = <FQ1>;

        my $reverse1 = <FQ2>;
        my $reverse2 = <FQ2>;
        my $reverse3 = <FQ2>;
        my $reverse4 = <FQ2>;

my $F1 = 0;
my $R1 = 0;
my $F2 = 0;
my $R2 = 0;

        if($forward2 =~ /$hash{$k}{F}/){
            $forward2 = substr($forward2, $+[0], 1000);
            $forward4 = substr($forward4, $+[0], 1000);
            $F1 = 1;
$fq1_F ++;
$fq1 ++;
            print LG "$k\_F\t$forward1";
        }

        if($reverse2 =~ /$hash{$k}{R}/){
            $reverse2 = substr($reverse2, $+[0], 1000);
            $reverse4 = substr($reverse4, $+[0], 1000);
            $R1 = 1;
$fq2_R ++;
$fq2 ++;
            print LG "$k\_R\t$reverse1";
        }

        if(($F1 == 1)&&($R1 == 1)&&(length($forward2) > 10)&&(length($reverse2) > 10)){
            print OT1 "$forward1$forward2$forward3$forward4";
            print OT2 "$reverse1$reverse2$reverse3$reverse4";
$pd ++;
        }

        if($forward2 =~ /$hash{$k}{R}/){
            $forward2 = substr($forward2, $+[0], 1000);
            $forward4 = substr($forward4, $+[0], 1000);
            $F2 = 1;
$fq1_R ++;
$fq1 ++;
            print LG "$k\_R\t$forward1";
        }

        if($reverse2 =~ /$hash{$k}{F}/){
            my $end = $+[0] + 1;
            $reverse2 = substr($reverse2, $+[0], 1000);
            $reverse4 = substr($reverse4, $+[0], 1000);
            $R2 = 1;
```

```perl
#!/usr/bin/perl -w

# add-abundance.pl

use strict;

die "Usage: perl $0 [.fa] [summary.uc] [length cutoff Min:Max] [out.fa]\n" unless (@ARGV == 4);
open (FA, $ARGV[0]) or die "$ARGV[0] $!\n";
open (AB, $ARGV[1]) or die "$ARGV[1] $!\n";
open OT, ">$ARGV[3]" or die "$ARGV[3] $!\n";

my $len = $ARGV[2];
my @len = split /\:/, $len;
my $min = $len[0];
my $max = $len[1];

my %hash;
while(<AB>){
    chomp;
    my @line = split;
    $hash{$line[0]} = $line[1];
}
close AB;

$/ = ">";
<FA>;

if ($ARGV[2] eq "NA"){
    while(<FA>){
        chomp;
        my @line = split /\n+/;
        my $name = shift @line;
        my $seq = join "", @line;
        print OT ">$name;size=$hash{$name};\n$seq\n";
    }
}else{
    while(<FA>){
        chomp;
        my @line = split /\n+/;
        my $name = shift @line;
        my $seq = join "", @line;
        my $length = length $seq;
        if(($length >= $min)&&($length <= $max)){
            print OT ">$name;size=$hash{$name};\n$seq\n";
        }else{
            print ">$name;size=$hash{$name};\t$length is removed\n";
        }
    }
}

close FA;
close OT;
print "DONE!\n";
```

```
#!/usr/bin/env python2.7

# evaluate-blast.py

import sys, csv
from sets import Set
import resolvefunctions as rf
# from resolvefunctions import find_best_candidates, resolve_ambiguity, taxtreedata

blastfile = 'blast/all-asvs.blast'
taxtreefile = 'rsrc/taxonomy-trees.txt'
seqlengthfile = 'rsrc/asv-sequence-lengths.txt'
mainoutfile = 'blast/all-asvs.blast.evaluated'
ambigreportfile = 'blast/ambiguity-report.txt'
moresimpleranks = 'SK K P C O F G S subS'.split()
allsimpleranknames = "Superkingdom Kingdom Phylum Class Order Family Genus Species Subspecies".split()
allsimplerankdict = dict(zip(moresimpleranks, allsimpleranknames))
ranknames = "no_rank superkingdom kingdom subkingdom superphylum phylum subphylum superclass class subclass infraclass superorder order suborder parvorder
infraorder cohort superfamily family subfamily tribe subtribe genus subgenus species species_group species_subgroup subspecies varietas forma".split()
rankabbrs = "nr SK K subK superP P subP superC C subC infraC superO O subO parvO infraO cohort superF F subF tribe subtribe G subG S Sgroup Ssubgroup subS
varietas forma".split()

def evaluate_blastdata(otuname):

    reasons = []

    # b is all the BLAST hits for this OTU. reportline starts out as the top one of the BLAST hits, and gets more info added to it.
    b = expandedblastdata[otuname]
    taxawithhighestbitscore = ';'.join([s['Taxon Acc'] for s in b]).split(';')  # Full list of ambiguous accessions, for later counting
    ambiguitycount = len(Set(taxawithhighestbitscore))  # Multiple top bitscores are the same, but for different organisms

    if ambiguitycount == 1:
        # If it's unambiguous, just set the additional fields and go on.
        ambiguitydata = []
        ambigstatus = "unambiguous"
        reportline = b[0]      # There's only one, so just de-listify it.
        reporttaxa = rf.split_taxonomy(reportline['Taxon Acc'])   # Includes several treatments of the total taxon tree
        reportline['Top Taxon'], reportline['Lower Taxa'] = reporttaxa[0:2]
        reportline['Tupled Taxa'] = reporttaxa[2]
        reportline['Taxon Level'] = rf.get_taxinfo(reportline['Taxon Acc'])[-1][0]
        for rank in allsimplerankdict:    # For each, e.g., 'K', make a key e.g. 'Kingdom' in reportline with the actual taxon value
            reportline[allsimplerankdict[rank]] = reporttaxa[2][rank]
    else:
        # Otherwise, do ambiguity resolution and the possible line-surgery that entails.
        ambiguitydata = [line for line in b if line['Taxon Acc'] in list(taxawithhighestbitscore)]
        bestaccs, diversities = rf.find_best_candidates(ambiguitydata, rf.simpleranks, rf.taxtreedata)
        bestlines = [line for line in b if line['Taxon Acc'] in bestaccs]
        resolution = rf.resolve_ambiguity(otuname, ambiguitydata, bestaccs, diversities, rf.taxtreedata)
        # resolution contains these fields: lastcommon, divergencerank, divergencenum, counts, finaltaxonomy, bestaccs

        # Get the best info from the resolution, and use that to select a representative line.
        # If lines have the same bitscore but different details, take one of the ones with the highest percent identity.
        level, finalacc = resolution['lastcommon'].split(':')[0:2]
        finalname = ':'.join(resolution['lastcommon'].split(':')[2:])  # For those few annoying names with colons in them.

        matchinglines = [line for line in b if line['Taxon Acc'] == finalacc]

        if matchinglines:
            # If all good lines were the same species, use the actual lines from BLAST.
            ambigstatus = "resolved to species"
            reportline = sorted(matchinglines, key=lambda x: x['Pct Ident'])[-1].copy()
            reportline['Taxon Level'] = rf.get_taxinfo(reportline['Taxon Acc'], matchinglines[0]['Taxon Acc'])[-1][0]
        else:
            # Otherwise, take the scores from the set of (used) lines as a whole, and blank other line-specific fields.
            reportline = sorted([line for line in bestlines], key=lambda x: x['Pct Ident'])[-1].copy()
            reportline['Taxon Level'] = level
            reportline['Taxon Acc'] = finalacc
            reportline['Sci Name'] = finalname
            reportline['Common Name'] = finalname
            if level not in ['nr', 'SK', 'X']:
                reportline['GenBankGI'] = 'multiple'
                reportline['GenBankNucl'] = 'multiple'
                if rf.blastnamedata:
                    try:
                        reportline['Taxon Category'] = rf.blastnamedata[finalname]
                    except KeyError:
                        reportline['Taxon Category'] = 'unknown'
                else:
                    reportline['Taxon Category'] = 'needs lookup'
                ambigstatus = "resolved to consensus"
            else:
                reportline['GenBankGI'] = 'None'
                reportline['GenBankNucl'] = 'None'
                reportline['Taxon Category'] = 'None'
                ambigstatus = "unresolved"

        examplehierarchy = rf.get_taxinfo(bestlines[0]['Taxon Acc'], preservespecies=True)
        reporttaxa = rf.split_taxonomy(reportline['Taxon Acc'])   # Includes several treatments of the total taxon tree
        reportline['Top Taxon'], reportline['Lower Taxa'] = reporttaxa[0:2]
        for rank in allsimplerankdict:    # For each, e.g., 'K', make a key e.g. 'Kingdom' in reportline with the actual taxon value
            reportline[allsimplerankdict[rank]] = reporttaxa[2][rank]

        for maxfield in ['Pct Ident', 'e-Value']:
            reportline[maxfield] = max(line[maxfield] for line in bestlines)

        tempambigdata = []
        for a in ambiguitydata:
            if a['Taxon Acc'] in bestaccs:
                a['Used'] = '*'
            else:
                a['Used'] = "x"
            tempambigdata.append(a)

        ambiguitydata = [x for x in tempambigdata]

    # Below this are things you do to the final line, once you've resolved any ambiguity.
    reportline['Ambiguity'] = ambigstatus
    reportline['Seq Length'] = seqlengthdata[otuname]
```

```
    if float(reportline['Pct Ident']) < 90:
        reasons.append('pctident < 90')
    elif float(reportline['Pct Ident']) < 97:
        reasons.append('pctident 90-97')

    if float(reportline['Bitscore']) < 100: reasons.append('bitscore < 100')

    if reasons:
        reportline['BLAST Filter Failures'] = '; '.join(reasons)
    else:
        reportline['BLAST Filter Failures'] = 'PASS'

    # Final tally into a single status

    reportline['Passing Status'] = ''

    if reportline['BLAST Filter Failures'] == 'PASS':
        reportline['Passing Status'] = 'pass'
    elif reportline['BLAST Filter Failures'] == 'pctident 90-97':
        # If pctident between 90 and 97 is the *only* failure
        reportline['Passing Status'] = 'near-pass'
    else:
        reportline['Passing Status'] = 'failed'

    return reportline, ambiguitydata


# Main program flow starts here

blastdata = {}

# Ingest sequence length reference
seqlengthdata = {}
s = open(seqlengthfile)
lines = [x.strip().split('\t') for x in s.readlines()]
for line in lines:
    seqlengthdata[line[0]] = line[1]
s.close()

# Read basic BLAST data, including setting headers from first line.
f = open(blastfile)
blastfields = f.readline().strip().split('\t')
line = f.readline().strip().split('\t')

rankfields = allsimpleranknames
extrafields = ['Seq Length', 'Taxon Level', 'Top Taxon', 'Lower Taxa', 'Ambiguity', 'BLAST Filter Failures', 'Passing Status']
semicolonfields = ['Taxon Acc', 'Sci Name', 'Common Name']   # Fields that may contain semicolons in cases of identical reference sequences
outheaders = blastfields + rankfields + extrafields
outheaders.remove('Len Ident')
outheaders.remove('Score')

# Generate initial data structure, "blastdata"
while line != ['']:
    thisotu = line[0]
    otuname = thisotu.split(';')[0]
    blastdata[otuname] = []
    topbitscore = line[11]
    # Iterate over lines that match this OTU name and share the top score, and add each such line to the overall dictionary as its own set of named fields.
    while line[0] == thisotu:
        linedict = dict(zip(blastfields, line))
        if line[11] == topbitscore:
            blastdata[otuname].append(linedict)
        # Keep iterating inside the "while line[0]" loop, until seeing a different OTU or an inferior bitscore breaks us out to the "while line" loop.
        line = f.readline().strip().split('\t')

f.close()

# Generate initial version of final data structure, "expandedblastdata"
expandedblastdata = {}

for p in blastdata:                # p is an OTU ID
    expandedblastdata[p] = []      # This will be a list of dictionaries
    for b in blastdata[p]:         # b is a single BLAST result's dictionary
        if ';' not in b['Taxon Acc']:    # For most lines, just add them to the new list of dictionaries.
            expandedblastdata[p].append(b)
        else:                            # But if they have internal semicolons, make a separate dictionary for each taxid.
            splitfieldlists = {}
            threefieldvalues = []
            for vfield in semicolonfields:
                splitfieldlists[vfield] = b[vfield].split(';')
            for i in range(0, len(splitfieldlists['Taxon Acc'])):
                newdict = b.copy()       # Make a copy of the base dictionary, but....
                for vfield in semicolonfields:     # ...replace the semicolon-containing fields with specific instances.
                    newdict[vfield] = splitfieldlists[vfield][i]
                expandedblastdata[p].append(newdict)

f = open(ambigreportfile, 'w')
m = open(mainoutfile, 'w')

m.write("%s\n" % '\t'.join(outheaders))

ambigfields = [x for x in blastfields]
ambigfields.insert(4, 'Used')
ambigfields.append('Top Taxon')
ambigfields.append('Lower Taxa')
f.write('%s\n' % '\t'.join(ambigfields))

# Now we can actually process them.
asvsinorder = sorted(expandedblastdata.keys(), key=lambda(x): int(x[3:]))
for p in asvsinorder:
    outline = []
    e, a = evaluate_blastdata(p)
    m.write("%s\n" % ('\t'.join([str(e[field]) for field in outheaders])))
    if a:
        f.write('.\n')  # Spacer line in output for readability
        for line in a:
            taxinfo = rf.get_taxinfo(line['Taxon Acc'])
            line['Top Taxon'] = taxinfo[0][2]
            line['Lower Taxa'] = '/'.join([x[2] for x in taxinfo[1:]])
```

```
            f.write("%s\n" % ('\t'.join([str(line[field]) for field in ambigfields])))

f.close()
m.close()
```

```
#!/usr/bin/env python2.7

# integrate-otus-and-samples.py

import sys, os, csv
from math import log
from collections import defaultdict
from sets import Set

import argparse

parser = argparse.ArgumentParser(description='Main eDNA data integration script. Currently hardcoded for UNDERC.')
parser.add_argument('-t', '--types', action="store", dest="inclusiontypes", default='p', help="Comma-separated types of OTUs to include. p=pass, np=near-pass,
f=failed.")
parser.add_argument('-o', '--outprefix', action="store", dest="outprefix", default='cumulative', help="Prefix for output files to be created.")
parser.add_argument('-d', '--basedir', action="store", dest="basedir", default='.', help="Path to folder containing data and rsrc directory.")
parser.add_argument('-b', '--blastfile', action="store", dest="blastfile", default='blast/AllOTUs.blast.evaluated', help="Path to evaluated BLAST results,
relative to basedir.")
args = parser.parse_args()

incllookup = {'p': 'pass', 'np': 'near-pass', 'f': 'failed'}
incltypes = []
for incltype in args.inclusiontypes.split(','):
    if incltype in ['p', 'np', 'f']:
        incltypes.append(incllookup[incltype])
    else:
        print "Inclusion types must be a combination of: p=pass, np=near-pass, f=failed."
        sys.exit(2)

outprefix = args.outprefix
basedir = args.basedir

blastfile = os.path.join(basedir, args.blastfile)
seqlengthfile = os.path.join(basedir, 'rsrc/otu-sequence-lengths.txt')
samplebreakoutfile = os.path.join(basedir, 'reports/sorted-samples-and-otus-by-total.txt')
samplenamesfile = '/projects/spe1/edna/metadata/sample-names-and-order.txt'

# These don't all actually need to be opened this early, but I was relocating the naming.
# taxexcloutfile = open(outprefix + '.0.taxon-exclusion', 'w')
otuoutfile = open(outprefix + '.1.otus', 'w')
orgoutfile = open(outprefix + '.2.primer-orgs', 'w')
trueorgoutfile = open(outprefix + '.3.organisms', 'w')
passorgoutfile = open(outprefix + '.4.organisms.passing', 'w')

otuoutputfields = ['Primer', 'OTU_ID', 'Primer_OTU', 'GenBankGI', 'GenBankNucl', 'Taxon Acc', 'Taxon Category', 'Sci Name', 'Common Name', 'Taxon Level', 'Top
Taxon', 'Lower Taxa', 'Seq Length', 'Pct Ident', 'e-Value', 'Bitscore', 'Ambiguity', 'BLAST Filter Failures', 'Passing Status', 'Org Passes', 'Org Near-Passes',
'Org Failures', 'Total Reads', 'OTU Sample Prs', 'OTU Lake Prs']
orgcopyfields = ['Primer', 'Taxon Acc', 'Taxon Category', 'Sci Name', 'Common Name', 'Taxon Level', 'Top Taxon', 'Lower Taxa']
orgoutputfields = orgcopyfields + ['OTU Passes', 'OTU Near-Passes', 'OTU Failures', 'Total Reads', 'Retained Reads', 'Org Sample Prs', 'Org Lake Prs']
# trueorgoutputfields will be defined later.

# This is the same as orgoutputfields at this point, except without OTU-specific primer and GenBank info.
trueorgfields = [x for x in orgoutputfields if x not in ['Primer', 'GenBankGI', 'GenBankNucl']]

def simple_reader(filename):
    outdict = {}
    f = open(filename)
    lines = [x.strip().split('\t') for x in f.readlines()]
    for line in lines:
        outdict[line[0]] = line[1:]
    f.close()
    return outdict

def csv_reader(filename, keyfield, fieldnames=None):
    outdict = {}
    f = open(filename)
    if fieldnames == None:
        fieldnames = f.readline().strip().split('\t')
    templines = f.readlines()
    tempreader = csv.DictReader(templines, delimiter="\t", fieldnames=fieldnames)
    for linedict in tempreader:
        linekey = linedict[keyfield].split(';')[0]
        outdict[linekey] = linedict
    f.close()
    return outdict


##########  Main program flow  ###########

mergedotudata = {}
pfadict = defaultdict(list)  # A dictionary that assumes, in the absence of other information, that all keys exist and have values of an empty list; i.e.,
pfadict[x] = [] for all possible x.

seqlengthdata = simple_reader(seqlengthfile)
blastdata = csv_reader(blastfile, 'Primer_OTU_Size')
sampledata = csv_reader(samplebreakoutfile, 'OTU_ID')
samplenamedata = csv_reader(samplenamesfile, 'Sample ID')

sampleidsbylakedict = defaultdict(list)
for k in samplenamedata:
    sampleidsbylakedict[samplenamedata[k]['Lake']].append(k)

# The short names (e.g., 32-1) and the long names (e.g., Red1S-32-1) in order.
orderedsamplekeys = sorted([(int(samplenamedata[x]['OutSort']), samplenamedata[x]['Sample ID'], samplenamedata[x]['Long Name']) for x in samplenamedata])
sampleidorder = [y[1] for y in orderedsamplekeys]
samplenameorder = [y[2] for y in orderedsamplekeys]

otuoutputfields.extend(samplenameorder)
orgoutputfields.extend(samplenameorder)

# Loop over all OTUs, in the BLAST context, and copy the relevant information to the new "mergedotudata" dictionary.
# Generates *both* mergedotudict and pfadict
for b in blastdata.keys():
    # Make a copy of the data from this row. m is, e.g., {'Primer_OTU_Size': 'Haz18S_OTU_4;size=12345', 'Common Name': 'bluegill' ... }
    m = blastdata[b].copy()
    # Parse the long form of Primer_OTU_Size into separate fields, and add them individually to m.
    primer, otuid, x, size = m['Primer_OTU_Size'].replace('_', ' ', 1).replace(';', ' ').replace('=', ' ').split()
    primer_otu = primer + '_' + otuid
    m['Primer'] = primer
```

```
    m['OTU_ID'] = otuid
    m['Total Reads'] = int(size)

    # Construct a blank sample record if one doesn't exist, revert the Primer_OTU field, and index by the long name.
    # If k is, e.g., '32-10', samplenamedata[k] is 'Red4D-32-10'.
    for k in sampleidorder:
        longname = samplenamedata[k]['Long Name']
        datastring = sampledata[primer_otu][k]
        m[longname] = datastring
        m['Primer_OTU'] = primer_otu

    m['otu_sam_prs_list'] = [k for k in sampleidorder if sampledata[primer_otu][k] != "."]
    m['OTU Sample Prs'] = len(m['otu_sam_prs_list'])
    numlakes = 0
    m['otu_lake_prs_list'] = []

    for lake in sampleidsbylakedict:
        lakesamples = sampleidsbylakedict[lake]
        presencesinthislake = [sample for sample in lakesamples if sample in sampledata[primer_otu] and sampledata[primer_otu][sample] != "."]
        if len(presencesinthislake) and lake != ".":
            m['otu_lake_prs_list'].append(lake)

    m['OTU Lake Prs'] = len(m['otu_lake_prs_list'])

    # Bring in the sequence length from that source dictionary
    m['Seq Length'] = seqlengthdata[b][0]

    # Add this OTU's info to the overall info for its organism
    if m['BLAST Filter Failures'] == 'PASS':
        pfadict[m['Taxon Acc']].append('pass')
    else:
        # If the *only* thing wrong with the OTU is that its pctident is slightly below 97, make it a near-pass.
        if m['BLAST Filter Failures'] == 'pctident 90-97':
            pfadict[m['Taxon Acc']].append('near-pass')
        else:
            pfadict[m['Taxon Acc']].append('fail')

    mergedotudata[primer_otu] = m

# Now fold the by-organism totals back into the individual OTUs, and also start an independent organism dictionary.
# This is a nested defaultdict, so that we can append directly to the empty list orgdata[x][y] without explicity having to create orgdata[x] or orgdata[x][y]
first.
orgdata = defaultdict(lambda: defaultdict(list))

for primer_otu, v in mergedotudata.items():
    # For each OTU, write the organism pass/fail numbers into both output dictionaries.
    primer_org = (v['Primer'], v['Taxon Acc'])
    taxid = v['Taxon Acc']
    p = pfadict[taxid].count('pass')
    np = pfadict[taxid].count('near-pass')
    f = pfadict[taxid].count('fail')
    mergedotudata[primer_otu]['Org Passes'], mergedotudata[primer_otu]['Org Near-Passes'], mergedotudata[primer_otu]['Org Failures'] = p, np, f
    orgdata[primer_org]['OTU Passes'], orgdata[primer_org]['OTU Near-Passes'], orgdata[primer_org]['OTU Failures'] = p, np, f

    # orgcopyfields is a list of fields that are the same for each OTU belonging to a given organism. Copy those values (repeatedly, alas) into orgdata.
    for fld in orgcopyfields:
        orgdata[primer_org][fld] = v[fld]

    # Now add to the overall primer/organism record, for each OTU and each sample, by keeping
    #   (1) running lists of all the OTU-specific sample and lake presences and
    #   (2) a running list of the read totals from each OTU.
    #   (3) a running list of all OTUs' sample data grouped by primer and organism.
    # ...but ONLY for types (p, np, a, f) that have been requested.
    # This is easy because of the defaultdicts; any orgdata[primer_org][x] starts out as a blank list.

    if mergedotudata[primer_otu]['Passing Status'] in incltypes:
        orgdata[primer_org]['org_sam_prs_list'].extend(mergedotudata[primer_otu]['otu_sam_prs_list'])
        orgdata[primer_org]['org_lake_prs_list'].extend(mergedotudata[primer_otu]['otu_lake_prs_list'])
        orgdata[primer_org]['retainedreadslist'].append(mergedotudata[primer_otu]['Total Reads'])
        for sam in samplenameorder:
            # This starts out making a list of values, because that's the best way to handle it with the defaultdict. We'll sum them later.
            orgdata[primer_org][sam].append(mergedotudata[primer_otu][sam])

    orgdata[primer_org]['Total Reads'].append(mergedotudata[primer_otu]['Total Reads'])

# Now tidy up the orgdata dictionary by converting the lists of sample info to summary stats.
for primer_org in orgdata:
    orgdata[primer_org]['Org Sample Prs'] = len(Set(orgdata[primer_org]['org_sam_prs_list']))
    orgdata[primer_org]['org_lake_prs_list'] = list(Set(orgdata[primer_org]['org_lake_prs_list']))
    orgdata[primer_org]['Org Lake Prs'] = len(orgdata[primer_org]['org_lake_prs_list'])
    orgdata[primer_org]['Total Reads'] = sum([int(s) for s in orgdata[primer_org]['Total Reads'] if str(s).isdigit()])
    orgdata[primer_org]['Retained Reads'] = sum([int(s) for s in orgdata[primer_org]['retainedreadslist'] if str(s).isdigit()])
    for sam in samplenameorder:
        # Here we're summing all the OTUs from a given primer, as long as they made the p/np/a/f list.
        orgdata[primer_org][sam] = sum([int(s) for s in orgdata[primer_org][sam] if str(s).isdigit()])

# Finally, collapse orgdata to be truly by-organism. At this stage, we *max* the reads across primers, rather than summing.
trueorgdata = defaultdict(lambda: defaultdict(list))

for primer_org, v in orgdata.items():
    taxid = primer_org[1]
    for fld in trueorgfields:
        # Copy most org fields into the corresponding trueorg field
        trueorgdata[taxid][fld] = v[fld]
    for sam in samplenameorder:
        # Makes a list of value strings to take the maximum of later.
        trueorgdata[taxid][sam].append(v[sam])
    trueorgdata[taxid]['totalreadslist'].append(v['Total Reads'])
    trueorgdata[taxid]['retainedreadslist'].append(v['Retained Reads'])

# Now tidy up the trueorgdata dictionary by converting the lists of sample info to summary stats.
for taxid in trueorgdata:
    t = trueorgdata[taxid]
    for sam in samplenameorder:
        t[sam] = max([int(s) for s in t[sam] if str(s).isdigit()])
        if t[sam] != 0:
            t['org_sam_prs_list'].append(sam)
    t['Org Sample Prs'] = len(t['org_sam_prs_list'])
```

```
    for lake in sampleidsbylakedict:
        lakesamples = sampleidsbylakedict[lake]
        presencesinthislake = [samplename for samplename in [samplenameorder[sampleidorder.index(sampleid)] for sampleid in lakesamples] if samplename in
t['org_sam_prs_list']]
        if len(presencesinthislake) and lake != ".":
            t['org_lake_prs_list'].append(lake)

    # Sum the reads across samples as normal.
    t['Total Reads'] = sum(t['totalreadslist'])
    t['Retained Reads'] = sum(t['retainedreadslist'])
    t['Final Reads'] = sum([t[r] for r in samplenameorder])

# Munge each data set and write to its output file.

outstrings = {}
outstrings['header'] = '\t'.join(otuoutputfields)

sortfields = ['Total Reads', 'Lower Taxa', 'Primer_OTU']
sorttuples = []

for otuid in mergedotudata:
    o = mergedotudata[otuid]
    outline = []
    for f in otuoutputfields:
        outline.append(o[f])
    sortvalues = [o[f] for f in sortfields]
    sorttuples.append(tuple(sortvalues))
    outstrings[otuid] = '\t'.join([str(x) for x in outline])

sorttuples.sort(reverse=True)
otuoutfile.write(outstrings['header'] + '\n')

for t in sorttuples:
    otuid = t[2]
    otuoutfile.write(outstrings[otuid] + '\n')

otuoutfile.close()

orgoutfile.write('\t'.join(orgoutputfields) + '\n')
for k, v in orgdata.items():
    orgoutfile.write('\t'.join([str(v[f]) for f in orgoutputfields]) + '\n')

orgoutfile.close()

sortfields = ['Top Taxon', 'Lower Taxa', 'Taxon Acc']
sorttuples = []

# And NOW acknowledge that the samples are part of trueorgfields, so we can output them.
# trueorgfields.insert(trueorgfields.index('Org Sample Prs'), 'Total Reads')
# trueorgfields.insert(trueorgfields.index('Org Sample Prs'), 'Retained Reads')
trueorgfields.insert(trueorgfields.index('Org Sample Prs'), 'Final Reads')
trueorgfields.extend(samplenameorder)

passingorgs = []

for org in trueorgdata:
    o = trueorgdata[org]
    outline = []
    for f in trueorgfields:
        outline.append(o[f])
    sortvalues = [o[f] for f in sortfields]
    sorttuples.append(tuple(sortvalues))
    outstrings[org] = '\t'.join([str(x) for x in outline])
    if o['OTU Passes'] > 0:
        passingorgs.append(org)

sorttuples.sort()

trueorgoutfile.write('\t'.join(trueorgfields) + '\n')
passorgoutfile.write('\t'.join(trueorgfields) + '\n')

for t in sorttuples:
    org = t[2]
    trueorgoutfile.write(outstrings[org] + '\n')
    if org in passingorgs:
        passorgoutfile.write(outstrings[org] + '\n')
```

```perl
#!/usr/bin/perl -w

# rename_4.0.pl

use strict;

die "Usage: perl $0 [.fa] [prefix of reads] [out.fa]\n" unless (@ARGV == 3);
open (FA, $ARGV[0]) or die "$ARGV[0] $!\n";
open OUT, ">$ARGV[2]" or die "$ARGV[2] $!\n";
open CAST, ">$ARGV[2].cast" or die "$ARGV[2] $!\n";
my $pre = $ARGV[1];
$/=">";
my $seq;
my @raw;
my $c;
if($pre eq 0){
        $c = "00000000000000000001";
}else{
        $c = 1;
}

my $null = <FA>;
while(<FA>){
    chomp;
    s/\r//g;
        @raw = split /\n/;
        my $name = shift @raw;
        my @sampleinfo = split /_/,$name;
        my $runandsample = shift @sampleinfo;
        my $primer = shift @sampleinfo;
        my @runnumarr = split /-/,$runandsample;
        my $runnum = shift @runnumarr;
        $seq = join "", @raw;
    $seq =~ tr/[a-z]/[A-Z]/;
        print OUT ">$primer\_$pre\_$c\n$seq\n";
        print CAST "$runnum\t$primer\t$name\t$pre\_$c\n";
        $c++;
}

close FA;
close OUT;
close CAST;
print "DONE!\n";
```

```
#!/usr/bin/env python2.7

# resolvefunction.py

import sys, csv
from sets import Set

blastfile = 'blast/AllOTUs.blast.evaluated'
blastnamefile = 'rsrc/extra-blastnames.txt'
ambigreportfile = 'blast/ambiguity-report.txt'
taxtreefile = 'rsrc/taxonomy-trees.txt'

ranknames = "no_rank superkingdom kingdom subkingdom superphylum phylum subphylum superclass class subclass infraclass superorder order suborder parvorder
infraorder cohort superfamily family subfamily tribe subtribe genus subgenus species species_group species_subgroup subspecies varietas forma".split()
rankabbrs = "nr SK K subK superP P subP superC C subC infraC superO O subO parvO infraO cohort superF F subF tribe subtribe G subG S Sgroup Ssubgroup subS
varietas forma".split()
ranklist = dict(zip([x.replace('_', ' ') for x in ranknames], rankabbrs))
simpleranks = 'K P C O F G S'.split()
moresimpleranks = 'SK K P C O F G S subS'.split()

def ingest_taxonomy():
    # Ingest taxonomy reference
    taxtreedata = {}
    f = open(taxtreefile)
    lines = [x.strip().split('\t') for x in f.readlines()]
    for line in lines:
        taxtreedata[line[0]] = [tuple(x.split(':')) for x in line[1].split(';')]
    f.close()
    return taxtreedata

def ingest_blastnames():
    # Ingest "blast names" for intermediate taxa that didn't come with them
    blastnamedata = {}
    try:
        f = open(blastnamefile)
    except IOError:
        return {}
    lines = [x.strip().split('\t') for x in f.readlines()]
    for line in lines:
        blastnamedata[line[0]] = line[2]
    f.close()
    return blastnamedata

def get_taxinfo(taxacc, examplehierarchy=[], preservespecies=True):
    taxa = []
    try:
        t = taxtreedata[taxacc]
    except KeyError:
        if taxacc == 'None':
            return [('unknown', 'unknown', 'unknown')]
        try:
            exampletuple = [x for x in examplehierarchy if x[1] == taxacc][0]
        except IndexError:
            return [('unknown', 'unknown', 'unknown')]

        if taxacc == 'None':
            return [('unknown', 'unknown', 'unknown')]
        elif not examplehierarchy:
            return [('unknown', 'unknown', 'unknown')]

        taxonposition = examplehierarchy.index([x for x in examplehierarchy if x[1] == taxacc][0])
        t = examplehierarchy[0:taxonposition+1]

    tstrings = [':'.join(x) for x in t]
    tranks = [x[0] for x in t]
    tnames = [x[2] for x in t]

    for rank in moresimpleranks:
        if rank in tranks:
            taxa.append([x for x in t if x[0] == rank][0])
        else:
            rankpos = rankabbrs.index(rank)
            while rankpos > 0 and rank not in moresimpleranks:
                taxa.append([x for x in t if x[0] == rank][0])
                rankpos = rankpos - 1
                rank = rankabbrs[rankpos]

    if 'SK:2759:Eukaryota' in tstrings:
        # If the superkingdom is Eukaryota, and the kingdom is one of the big three, just ditch Eukaryota in favor of the kingdom.
        if list(set(['K:33208:Metazoa', 'K:4751:Fungi', 'K:33090:Viridiplantae']) & set(tstrings)):
            taxa.remove(('SK', '2759', 'Eukaryota'))                    # The tuple is what needs to be removed from taxa proper

    for weirdertoprank in ['nr:12908:unclassified sequences', 'nr:28384:other sequences']:
        # ...but if it's one of the weirder things, make sure we report that as the "kingdom".
        if weirdertoprank in tstrings:
            taxa.insert(0, tuple(weirdertoprank.split(':')))

    if 'S' in tranks and 'subS' in tranks and not preservespecies:
        # If species is just the first two words of subspecies, eliminate it for brevity.
        s = [x for x in t if x[0] == 'S'][0]
        ss = [x for x in t if x[0] == 'subS'][0]
        if s[2].split(' ')[0:2] == ss[2].split(' ')[0:2]:
            taxa.remove(s)

    return taxa       # A list of tuples in the style of taxtreedata

def split_taxonomy(taxacc, examplehierarchy=[]):
    taxa = get_taxinfo(taxacc, examplehierarchy)   # [('K', '1234', 'Martians'), ... ]
    taxadict = dict([(x[0], x[2]) for x in taxa])  # {'K': 'Martians', ... }
    for rank in moresimpleranks:
        if rank not in taxadict:
            taxadict[rank] = 'NA'
    taxanames = [x[2] for x in taxa]
    k = taxanames[0]
    t = taxanames[1:]
    if not t:
        t = ['No lower taxa in consensus']
    return k, '/'.join(t), taxadict
```

```
def find_best_candidates(linelist, simpleranks, taxtreedata):

    otu = linelist[0]['ASV']
    allaccs = list(set([l['Taxon Acc'] for l in linelist]))           # list of all taxaccs
    alluniqueaccs = list(set(allaccs))                                # ordered list of all unique taxaccs
    hierarchylist = [taxtreedata[acc] for acc in alluniqueaccs]       # same-ordered list of full hierarchy for each unique taxacc
    textlist = ['/'.join([x[2] for x in h]) for h in hierarchylist]   # same-ordered list of hierarchy name strings for each unique taxacc
    # candidatelists will end up being a list of total line sets, in increasing order of stringency (until it's reversed)
    candidatelists = []
    candidatelists.append(textlist)   # Initial copy of all the accessions' hierarchies
    candidatelists.append([x for x in candidatelists[-1] if 'environmental' not in x])
    candidatelists.append([x for x in candidatelists[-1] if not x.startswith('uncultured')])
    candidatelists.append([x for x in candidatelists[-1] if x.count('/') <= 1])
    candidatelists.append([x for x in candidatelists[-1] if 'uncultured' not in x])
    candidatelists.append([x for x in candidatelists[-1] if not x.startswith('fungal ')])
    candidatelists.append([x for x in candidatelists[-1] if not x.endswith(' sp.')])
    candidatelists.reverse()

    for candidateset in candidatelists:
        # Take the first set of candidates that has something in it.
        if len(candidateset):
            returnset = [x for x in candidateset]
            break

    bestaccs = [alluniqueaccs[n] for n in [textlist.index(x) for x in returnset]]
    uniquehierarchies = [taxtreedata[a] for a in bestaccs]

    # Create the diversity summary
    diversities = {}
    for rank in simpleranks:
        namelist = []
        allofthisrank = [[t[2] for t in h if t[0] == rank] for h in uniquehierarchies]
        for n in allofthisrank:
            if n:
                namelist.append(n[0])
            else:
                namelist.append('')
        diversities[rank] = len(set(namelist))

    # Special elimination for singleton ringers
    if diversities['G'] == 1 and diversities['S'] == 2:
        uniquespeciestuples = [[t for t in h if t[0] == 'S'][0] for h in [hierarchylist[n] for n in [textlist.index(x) for x in returnset]]]
        uniquespecies = [x[1] for x in uniquespeciestuples]
        relativespeciescounts = [len([l for l in linelist if l['Taxon Acc'] == a]) for a in uniquespecies]
        if min(relativespeciescounts) == 1 and max(relativespeciescounts) > 1:
            elimacc = uniquespecies[relativespeciescounts.index(1)]
            otheracc = uniquespecies[int(not relativespeciescounts.index(1))]
            elimaccname = [t[2] for t in taxtreedata[elimacc] if t[0] == 'S'][0]
            otheraccname = [t[2] for t in taxtreedata[otheracc] if t[0] == 'S'][0]
            if 'sp.' in elimaccname and 'sp.' not in otheraccname:
                bestaccs.remove(elimacc)
                diversities['S'] = 1

    return bestaccs, diversities

def resolve_ambiguity(otu, linelist, bestaccs, divs, taxtreedata):
    outdict = {}
    allaccs = [l['Taxon Acc'] for l in linelist]

    divergencerank, divergencenum = ('X', 0)

    samplehierarchy = taxtreedata[bestaccs[0]]
    consensushierarchy = []

    for rank in simpleranks:
        if divs[rank] > 1:
            # If the good hierarchies diverge at this point, record that and stop.
            divergencerank, divergencenum = (rank, divs[rank])
            break
        else:
            # Otherwise, add this level's info and carry on.
            thisrankstuple = [t for t in samplehierarchy if t[0] == rank]
            if len(thisrankstuple):
                consensushierarchy.append(thisrankstuple[0])

    counts = set([(a in bestaccs and "*" or "x", ['/'.join([x[2] for x in get_taxinfo(a)]) for l in linelist if l['Taxon Acc'] == a][0], allaccs.count(a)) for a
in allaccs])

    outdict = {}
    outdict['lastcommon'] = consensushierarchy and ':'.join(consensushierarchy[-1]) or None
    outdict['divergencerank'] = divergencerank
    outdict['divergencenum'] = str(divergencenum)
    outdict['counts'] = sorted(counts)
    outdict['finaltaxonomy'] = '/'.join([str(x[2]) for x in consensushierarchy if x[0] in simpleranks])
    outdict['bestaccs'] = bestaccs
    if outdict['divergencerank'] == 'K' or outdict['finaltaxonomy'] == '.' or outdict['lastcommon'] == None:
        outdict['lastcommon'] = 'X:None:None'
        outdict['finaltaxonomy'] = 'None/No consensus taxonomy'

    nosize = otu.split(';')[0]

    return outdict

taxtreedata = ingest_taxonomy()
blastnamedata = ingest_blastnames()
```

```perl
#!/usr/bin/perl -w

# unique.pl

use strict;

die "Usage: perl $0 [.fa] [out.fa]\n" unless (@ARGV == 2);
open (FA, $ARGV[0]) or die "$ARGV[0] $!\n";
open OUT, ">$ARGV[1]" or die "$ARGV[1] $!\n";
open OUT2, ">$ARGV[1].cast" or die "$ARGV[1].cast $!\n";
$/=">";
my (%abund, %name, %collapse);
<FA>;
while(<FA>){
    chomp;
    my @line = split /\n+/;
    my $name = shift @line;
    my $seq = join "", @line;

    my $length = length ($seq);
    next if ($length <32);

    if($abund{$seq}){
        $abund{$seq} ++;
        $collapse{$seq} .= $name."\n";
    }else{
        $abund{$seq} = 1;
        $name{$seq} = $name;
        $collapse{$seq} = $name."\n";
    }
}

foreach my $k (sort keys %abund){
    print OUT ">$name{$k};size=$abund{$k};\n$k\n";
    print OUT2 ">$name{$k};size=$abund{$k}\n$collapse{$k}";

}
print "Unique DONE!\n";
```