N N N N N N N N N N	a a hoan	hinary	troo as	and	rray								
2 Hean a	J a neap Moorithm	ns usin	n arrav	stor	iiray ane								
3. Heap s	sort		gunuy	5.01	uge								
1. Storin	g a heap	binary	tree as	s an a	array								
A heap is a	binary tre	e with c	ertain pi	ropert	ies.								
Assumptior	1: each ve	rtex (noo	de) in the	e bina	rv tree	has a v	/alue (a	a kev)	asso	ciate	d with	it	
								, ,					
To be a hea	p a binary	rree mu	ust have	the fo	bllowing	g prope	rties:						
property 1:	the paren	it's key i	s at leas	t as la	arge as	the key	ys of its	s child	lren				
property 2:	all depths	s, except	t possibl	y the	largest	, have t	he max	kimum	num	ber o	fnod	es	
property 3:	at the larg	gest dep	oth, any i	missir	ng node	es (leav	es) are	at the	e righ	t end			
							-						
From binar	v tree to a	rrav											
X		9	-> (1	1	9	8	>	•	m	00	00	6	2
5	6 8			Ð	5						00		9
									Weh	nave l	ost th	e poir	nters
									to ch	lldrer	n and	paren	its!
							find the		Iron a	nd/o	noro	nt of	2
Q: When a	binary tree	e is stor	ed as ar	array	/, how	can we	inna ine			anu/o	pare		a

Array indices and locations in binary tree. Increasing 4 3 4 5 6 7 6 7 6 7 1 1 1 1 1 1 1 1	corresponding array . 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15												
Example													
heap as binary tree corresponding array													
7 9 5 6	11 7 9 5 6 8												
For node whose index is x in array: LEFT(x) = $2x$, RIGHT(x) = $2x + 1$,	$PARENT(x) = \lfloor x/2 \rfloor$												
Assumption: heap array is A													
NOTES													
 key of root is always A[1] A heapsize is number of elements in hear 													
3. <i>A.length</i> must be at least <i>A.heapsize</i> 4. assume functions LEFT, RIGHT and PAR	ENT are available												

Consider swappin	ig key of <i>node1</i> with	its parent ke	y. Assum	e A[x] corre	esponds to	node1.
Using pointers			U	sing array		
1 4			(12)		1	_
-1 temp = node1.key 2 node1 key = node1	parent keu		1	temp = A[: $A[r] - A[I]$	x_{j} PARENT (x_{j})	
3 node1.parent.key =	= temp		3	A[x] = A[1] A[PAREN]	VT(x) = temp	
						= 🗆
O : Why store a be	an as an array?					
	sup do un anay.					
A: It is more efficient	ent both in terms of	running time	and storag	ge space to	o store a he	ap as an
anav rather than t	using a binary tree v	with pointers.				
	5					
2. Heap algorith	ms using array s	torage				
2. Heap algorith	ms using array s	torage				
2. Heap algorithi	ms using array s ows insertion of a n	torage ew node into	an existing	g heap		
2. Heap algorithi	ms using array s ows insertion of a n	torage ew node into	an existing	g heap		
2. Heap algorith	ms using array s	torage ew node into	an existing	g heap		
2. Heap algorithi	ms using array s ows insertion of a n	torage ew node into	an existing	g heap		
2. Heap algorithing HEAP-INSERT: all HEAP-INSERT: all HEAP-INSERT(<i>ha</i> input a heap new node that	ms using array s ows insertion of a ne eapRoot, node) whose root node is spe should be added to the	torage ew node into	an existing	g heap		
2. Heap algorithi HEAP-INSERT: all HEAP-INSERT: all HEAP-INSERT(he input a heap new node that if depth heigh	ms using array s ows insertion of a ne eapRoot, node) whose root node is spe should be added to the	torage ew node into cified by heapRo heap	an existing	g heap		
2. Heap algorithi	ms using array s ows insertion of a ne eapRoot, node) whose root node is spe should be added to the t is not yet full of le de as the rightmost lea	torage ew node into cified by heapRo heap aves then if at depth heigh	an existing ot and a	g heap	2	
2. Heap algorith HEAP-INSERT: all 1 HEAP-INSERT: all 2 input a heap 3 new node that 4 5 6 pinsert no else 8 height = height 9 insert no	ms using array s ows insertion of a ne eapRoot, node) whose root node is spe should be added to the t is not yet full of le de as the rightmost leas t+1 de as the first (leftmost	torage ew node into cified by heapRo heap vaves then of at depth heigh	an existing ot and a t th height	g heap		
2. Heap algorith HEAP-INSERT: all 1 HEAP-INSERT: all 2 input a heap 3 new node that 4 5 6 7 else 8 9 height = height 9 insert not end	ms using array s ows insertion of a new exapRoot, node) whose root node is spe should be added to the t is not yet full of le de as the rightmost lead t+1 de as the first (leftmost	torage ew node into cified by heapRo heap aves then af at depth heigh ost) leaf at dep	an existing ot and a t th height	g heap		
2. Heap algorithi HEAP-INSERT: all HEAP-INSERT: all HEAP-INSERT(ha input a heap new node that if depth height b insert no else height = height p insert no else height = height p insert no else height = height p insert no end	ms using array s ows insertion of a new capRoot, node) whose root node is spe should be added to the t is not yet full of lead t + 1 de as the first (leftmod insertion, node now ha ate up in the heap unit	torage ew node into cified by heapRo heap aves then of at depth heigh ost) leaf at dep is a parent. We til it finds its	an existing of and a t th height allow	g heap		
2. Heap algorith HEAP-INSERT: all 1 HEAP-INSERT: all 2 input a heap 3 new node that 4 if depth height 5 if depth height 6 b insert no else height = height 5 insert no end 1 2 /* After this	ms using array s ows insertion of a ne eapRoot, node) whose root node is spe should be added to the t is not yet full of le de as the rightmost lea t+1 de as the first (leftmo	torage ew node into cified by heapRo heap aves then of at depth heigh ost) leaf at dep	an existing of and a t th height	g heap		

	1 $HEAP_INSERT(A - bcm)$				
5	1 DEAT -INSERT (A, key) 2 input a heap which is stored as an array A and the key	of			
	2 input a heap which is stored as an array A and the key 2 new node that should be added to the beap	ora			
	$\frac{1}{4}$ here is a should be added to the heap				
	$\begin{aligned} 4 & l = A.heapsize + 1 \\ 5 & A[i] = h \end{aligned}$				
	$\begin{array}{c} 5 \\ c \end{array} \qquad A[i] = key \\ c \end{array}$				
	$6 \qquad A.heapsize = i$				
	7 while $(i > 1 \text{ and } A[i] > A[PARENT(i)])$				
	8 temp = A[i]				
	9 $A[i] = A[PARENT(i)]$				
1	0				
1	1 i = PARENT(i)				
1	2 end				
HEA	PIFY : for a binary tree (or subtree) for which some node <i>b</i>	x lacks pr	operty 1	, restores t	his
prop	erty for all nodes below node x thereby making a heap				
prop	ong for an nodeo below hodo x moloby maxing a nodp				
<u>.</u>		<u>_</u>			
1	HEAPIFY (heapRoot, node1)				
2	input a neap whose root node is specified by <i>heaphoot</i> and one particular node (node) of the heap				
3	/* All nodes in the heap except for possibly nodel				
5	have the heap properties. We allow <i>nodel</i> to trickle down to				
6	its correct location. */				
7					
8	node2 = NIL				
9	while $(node1 \neq node2)$			γ	b V
10	node2 = node1			\sim	
11	L = node1.teft, $R = node1.rightif (L exists and L key > node2 key) then$				
13	node2 = L			\sim	2
14	end		0	0	0
15	if $(R \text{ exists and } R.key > node2.key)$ then				
16	node2 = R				
17	end				
18	if $(node1 \neq node2)$ then				
	SwAP(node1, node2)		_		
20	end				
988 <u></u>	0000000	⇒			
1	HEAPIFY(A = i)	-			
2	input a heap which is stored as an array 4 and a location i				
3	whose key may be smaller than its children				
4	i = A heansize + 1				
5	j = 1 incorporation $j = 1$				
6	i = i				
7	$\int - \frac{1}{2} e^{-it} \frac{1}{2} \int \frac{1}{2} \frac{1}{2$				
8	if $(I \leq A \text{ heavisize and } A[I] > A[i])$ then				
0	i = I				
	end				
11	if $(B \le A \text{ hearsize and } A[R] > A[i])$ then				
19	i = R				
12	j = n		_		
13					
14	11 $(i \neq j)$ then				
15	temp = A[i]				
16	A[i] = A[j]				
17	A i = temp				
18	end				
18 19	end				

prop	peny in															
1	BUILDH	IEAP(hea	pRoot)													
2	input	a binar	y tree	whose	e root	node	is spe	cified	by heap	Root	5.0.0	6.0				firs
3	/* In of it:	e input s nodes.	As s	uch a	may 1a	es, er	operty xcept	for the	ny or a curren	it.	320	V RM		<'\		1
5	leaves	s, must	be ch	necked	and if	neces	ssary	modifie	d so th	at		\ <mark>`</mark>				• • •
6 7	the f	inal out	put is	a bii	nary ti	ree wh	ich is	a heap	•. */			1				در
8	for <i>i</i>	from her	ght - 1	to 0												
9	forI	Each int	ernal	node a	t dept	h <i>i</i> fro	m rigl	ht to l	eft			4	\ 5		ć	
10	end	EAPIFY (7	eapRoot	, noae	:)							4	5		0	
12	end															
1												,				
											260	Dug	tim	ł ł		
1	BUILD	HEAP(4)									·					
2						50 1 7							1	1.	1 1	
4	input	a bina	ry tre	e which	ch is a	stored	as an	array	A		1		13	14	151	b }
3	input /* It	a bina is assi	ry tre 1med t	e whic <mark>h</mark> at tl	ch is : he ent	stored ire ari	as an ray is	array used i	A in stori	ng the	1	12	3	4	5	6 }••
2 3 4	<mark>input</mark> ∕∗ It binar	a bina is assi y tree	ry tre 1med t elemen	e which hat th nts. */	ch is : he ent /	stored ire ari	as an ray is	array used i	A in stori	ng the	1	ld	3	14	5	6 }••
2 3 4 5	input /* It binar A.hea	is a bina is assure the transformation of transform	ry tre 1med t elemen <i>length</i>	e which hat that that the second seco	ch is : he ent /	stored ire ari	as an ray is	array used i	A in stori	ng the	1	2	3	4	5	6 }••
2 3 4 5 6 7	input /* It binar A.heap for i	a bina is assi y tree psize = A. from [(ry tre umed t elemen <i>length</i> A.heaps	c which hat the nts. */ ize/2)]	ch is s he ent / to 1	stored ire ari	as an ray is	array used i	A in stori	ng the		d	3	} 4	5	6 }••
2 3 4 5 6 7 8	input /* It binar A.heaj for i H	is a bina is assi y tree osize = A. from [(IEAPIFY(ry tre umed t elemen length A.heaps A, i)	e which hat th nts. */ ize/2)]	ch is s he ent / to 1	stored ire ari	as an ray is	array used i	A in stori	ng the		d	3	} 4	5	6 }••
	input /* It binar A.heaj for i H end	is a bina is assury tree psize = A. from [(IEAPIFY(ry tre imed t elemen <i>length</i> <i>A.heaps</i> <i>A</i> , <i>i</i>)	e which hat th hts. */ ize/2)]	ch is the ent / to 1	stored ire ari	as an ray is	array used i	A in stori	ng the	1	d	3	} 4	5	6 }••
	input /* It binar A.heap for i H end	is a bina is assury tree osize = A. from [(IEAPIFY(ry tre umed t elemen length A.heaps A, i)	e which hat the second	ch is the ent / to 1	stored ire ari	as an ray is	array used i	A in stori	ng the			3		5	6 }••
2 3 4 5 6 7 8 =	input /* It binar A.heaj for i H end	a bina is assi y tree from [(HEAPIFY(ry tre umed t elemen length A.heaps A, i)	e which hat the state of the st	ch is the ent / to 1	stored ire arr	as an ray is	array used i	A in stori	ng the			3		5	6 }
2 3 4 5 6 7 8 =	input /* It binar A.heaj for i H end	is a bina is assury tree osize = A. from [(IEAPIFY(ry tre umed t elemen length A.heaps A, i)	e which hat the second	ch is the ent	stored ire ari	as an ray is	array used i	A in stori	ng the			3		5	6 }
2 3 4 5 6 7 8 =	input /* It binar A.heaj for i H end	a bina is assi y tree bosize = A. from [(HEAPIFY(ry tre umed t elemen length A.heaps A, i)	e which hat the second	ch is she ent / to 1	stored ire ari	as an ray is	array used i	A in stori	ng the					5	6 }
2 3 4 5 6 7 8 =	input /* It binar A.heaj for i H end	a bina is assi y tree osize = A. from [(IEAPIFY(ry tre umed t elemen length A.heaps A, i)	e which hat the second	ch is the ent	irc ari	as an ray is	array used i	A in stori	ng the					5	6 }
2 3 4 5 6 7 8 =	input /* It binar A.heaj for i H end	a bina is assury tree osize = A. from [(IEAPIFY(ry tre umed t elemen length A.heaps A, i)	e which hat the second	ch is she ent / to 1	ire ari	as an ray is	array used i	<i>A</i> in stori	ng the					5	6 }
2 3 4 5 6 7 8 =	input /* It binar A.heaj for i H end	a bina is assi y tree osize = A. from [(IEAPIFY(ry tre imed t elemen length A.heaps A, i)	e which hat the second	ch is she ent / to 1	stored ire ari	as an ray is	array used i	A in stori	ng the					5	6 }
2 3 4 5 6 7 8 =	input /* It binar A.heaj for i H end	a bina is assi y tree osize = A. from [(IEAPIFY(ry tre umed t elemen length A.heaps A, i)	e which hat the second	ch is the ent	stored ire ari	as an ray is	array used i	A in stori	ng the					5	6 }
	input /* It binar A.heaj for i H end	TPACT	ry tre umed t elemen length A.heaps A, i)	e which hat the second	to 1	tho ro	as an ray is	array used i								
2 3 4 5 6 7 8 =	input /* It binar A.heaj for i H end AP-EXT maximu	TRACT	ry tre umed t elemen length A.heaps A, i)	e which hat the second	to 1	the rc	oot fro	array used i	A in stori	estores	the he	ap pr	operti	ies ar	5 nd ret	G }
3 4 5 6 7 8 =	input /* It binar A.heaj for i H end AP-EXT maximu	TRACT	ry tre umed t elemen length A.heaps A, i)	e which hat the second	to 1	the rc	oot fro	array used i	A in stori	estores	the he	ap pr	operti	ies ar	5	G }
2 3 4 5 6 7 8 =	input /* It binar A.heaj for i H end AP-EXT maximu	TRACT	ry tre umed t elemen length A.heaps A, i)	e which hat the second	to 1	the rc	as an ray is	array used i	A in stori eap, re	estores	the he	ap pr	operti	ies ar	5	G }
² 3 4 5 6 7 8 =	input /* It binar A.heaj for i H end AP-EXT maximu	TRACT	ry tre imed t elemen length A.heaps A, i)	e which hat the second	to 1	the rc	oot fro	array used i	A in stori eap, re	estores	the he	ap pr	operti	ies ar	5	c }••
2 3 4 5 6 7 8 =	input /* It binar A.heaj for i F end	realis assive the second secon	ry tre imed t elemen length A.heaps A, i)	e which hat the second	to 1	the rc	oot fro	om a he	A in stori eap, re	estores	the he	ap pr	operti	ies ar	5	G }
2 3 4 5 6 7 8 =	input /* It binar A.heaj for i P end AP-EXT maximu	TRACT	ry tre imed t elemen length A.heaps A, i)	e which hat the second	to 1	the rc	oot fro	array used i	A in stori	estores	the he	ap pr	operti	ies ar	15	C }

	HEAP-EXTRACI-MAX(heapRoot)
2	The react of the been is remered and replaced by the
ð	rightmost loaf at the lowest depth. After restoring the
+ 5	heap properties the key of the original rest is returned */
6	neap properties, the key of the original 700 is returned. */
7	max = heanBoot key
	\triangleright let node be the rightmost leaf at depth height
9	heanRoot key = node key
10	\triangleright remove <i>node</i> from the heap
11	if node was the only node at depth height
12	height = height - 1
13	end
14	HEAPIFY(heapRoot, heapRoot)
15	return max
1	HEAP-EXTRACT-MAX (A)
2	input a heap which is stored as an array A
3	
4	max = A[1]
5	n = A.heapsize
6	A[1] = A[n]
	A.neapsize = n - 1 $HEADIEV(A = 1)$
9	return max
3. He	eap sort
A hea	p can be used to produce a sorted array using procedure HEAPSORT.
1	$\operatorname{HEAPSORT}(A)$
2	input an array A which contains numbers that have no
	particular order
3	/* This procedure sorts the numbers in A from smallest to
	largest. It is assumed that all locations in A are used. */
5	BUILDHEAP(A)
6	n = A.length
7	for i from n to 2
8	temp = A[i]
9	A[i] = A[1]
10	A[1] = temp
11	A.heapsize = A.heapsize - 1
12	HEAPIFY(A = 1)
13	end
10	

Exam	ple									
Starti	ng array:	_A: 2	6 4 5	8 11	3					
stage	line	computat	tion or new	A						
1	5	11 8	4 5	6 2 3						
2	6, 7	n = 7, i =	= 7							
3	8,9,10,11	temp = 3	, A[7] = 11	, A[1] = 3, .	A. <mark>he</mark> apsiz	e = 6	38	4 5	6	2
4	12	8 6	4 5	3 2 11			2 6	4 5	3	8
5	7,8,9,10,1	1 i = 6, ter	np = 2, A[6]	[b] = 8, A[1]	= 2, A.he	apsize =	= 5			
6	12	6 5	4 2	3 8 11						
								— —		
NOTE 1. Hea 2. Hea	S apsort need	s no extra st	orage spac	e. me as Merg	esort's ru	ntime ef	ficiency.			
	·									

Täm	ä teos on	lisensoitu	Creative (Commo	ns Nimeä·	EiKaupa	llinen-				
EiMu <u>http:</u>	uutoksia 4. //creativec	0 Kansair commons.c	välinen -li org/license	senssil s/by-no	lä. Tarkas c-nd/4.0/.	tele liser	issiä os	oittee	ssa		
tekija	ä: Frank C	ameron									
Thia	work in lin		dor the Cr	ootive (Commons	Attributio	n				
Non	Commercia	al-NoDeriv	atives 4.0	Interna	ational Lic	ense. To	view a	сору	of		
this I	license, vi	sit <u>http://cr</u>	eativecom	mons.c	org/license	es/by-nc-	<u>nd/4.0/</u> .				
mad	e by Frank	Cameror	1							 	
6	20	3									
C											
_											