CSCE 500 Spring 2014
Homework #2 (Due on 2/20/2014)

1. For the following questions, assume that an array implementation of heap.

a. If you are adding one element at a time onto a heap that maintains min heap property, find the tight bound taken to add N elements to the min-heap. → add N elem.

b. Suppose these N elements are already in an array, not in any particular order. Find the tight bound to maintain the min-heap property for this array of N elements. → create heap from array

c. Assuming that the content of an array in problem b. is shown as 10,12,7,1,5,9,20,6,15. Show the content of the array after maintaining the min-heap property.

d. Obtain the time complexity of combining two min heaps of length N.

2. Solve for the recurrence relation of the following using the master theorem

a. $T(n)= 2T(n/2) + n^3$ $= \Theta(n^3)$

b. $T(n) = 16\,T(n/4) + n^2$ $= \Theta(n^2 \log n)$

c. $T(n) = 7\,T(n/2) + n^2$ $= \Theta(n^{\log_2 7})$

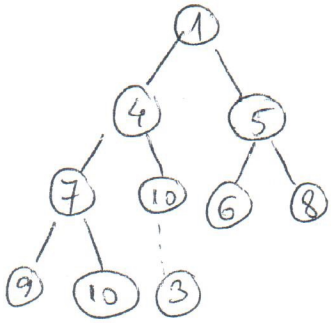
d. $T(n) = 2T(n/4) + n^{0.5}$ $= \Theta(\sqrt{n} \log n)$

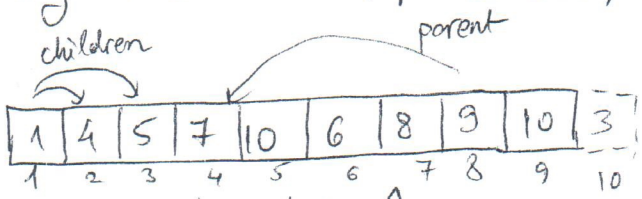
e. $T(n) = 3T(n/2) + n \log n$ $= \Theta(n^{\log_2 3})$

① a) Adding n elements to the heap

Suppose the heap has already P elements (P = constant)

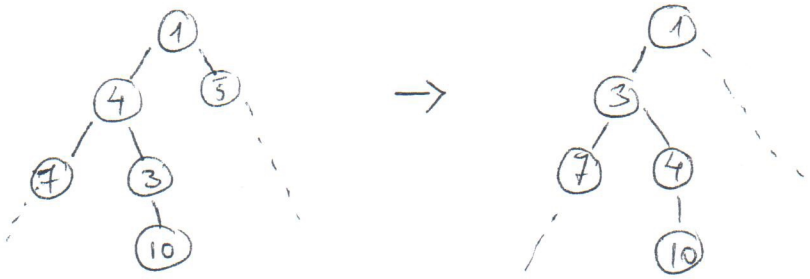
Example : P = 9





The array implementation ↑

Adding one element : X = 3 , it goes at the bottom-right of the heap (end of array) & exchanges with its parent until X < all its children :



If the final position is always the root node (worst case), it would take:

log P steps to add the first element

log (P+1) —"— 2nd elem.

⋮

log (P+N-1) to add the Nth element , assuming cost of swap is constant O(1)

In total $\log(P(P+1)\dots(P+N-1))$ steps $= \log \frac{(P+N-1)!}{(P-1)!} = \log(P+N-1)! - \log(P-1)!$ ↓ constant

From last Homework, I already proved that $\log n! = \Omega(n \log n)$

$\Rightarrow \log(P+N-1)! = \Omega((P+N-1)\log(P+N-1)) = \Omega(N \log N)$ as P = constant

$\Rightarrow$ The total cost for adding N elements in the worst case is $\Omega(N \log N)$

In the best case, each element for adding is greater than all the elements in the heap (the N elements come sorted DESC)

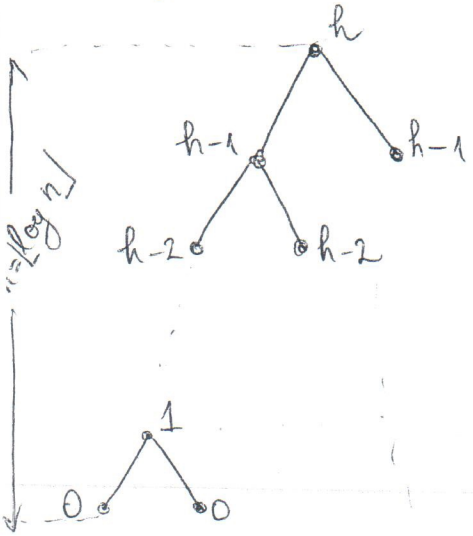
It takes O(1) to add one element (cost of comparation with its parent and addition at the bottom of the array) $\Rightarrow$ for N elements total cost is $O(N)$ or $\Omega(N)$.

$\Rightarrow$ Tight bound is between $\Omega(N)$ and $\Omega(N \log N)$ but we cannot determine exactly because of no information about the N elements.

b) ≡ Constructing a heap from an array of N elements — O(N)

We could follow the approach from point a), which takes $O(N \log N)$

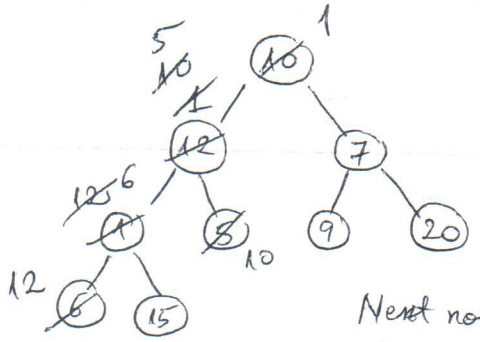
Otherwise, we can construct the heap tree as it is given in the array by respecting the heap shape & swap places.



For each node (position) $\lfloor \frac{N}{2} \rfloor$ which has children (starting with node $\lfloor \frac{N}{2} \rfloor$)

* swap the node with its lowest child
* go downwards of its swapped element to verify the min-heap property & apply the same algorithm until reaching level 0.

Example: c) 10, 12, 7, 1, 5, 9, 20, 6, 15



$N = 9$

$\lfloor \frac{N}{2} \rfloor = 4$ => element (1) is the first

(1) respects the property

Next node is (7), which respects

Next node is (12), which swaps with (1)

→ For (12) we go downwards and do the same thing

Swap (12) with (6) and stop.

Next node is (10), which swaps with (1)

→ For (10) we go downwards and swap (10) with (5)

(5) does not have children now. so stop.

The final heap is that whose array is now:

1, 5, 7, 6, 10, 9, 20, 12, 15

10	12	7	1	5	9	20	6	15
1	2	3	4	5	6	7	8	9

start

OK

10	12	7	1	5	9	20	6	15
1	2	3	4	5	6	7	8	9

current

OK

10	12	7	1	5	9	20	6	15
1	2	3	4	5	6	7	8	9

current

swap with pos. 4 (elem. (1))

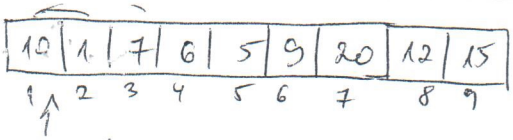
10	1	7	12	5	9	20	6	15
1	2	3	4	5	6	7	8	9

current — propagate downwards

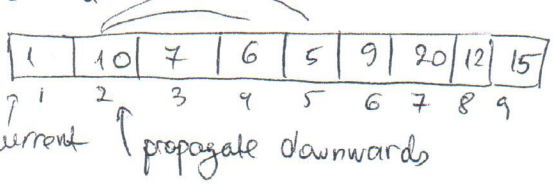
=>

10	1	7	6	5	9	20	12	15
1	2	3	4	5	6	7	8	9

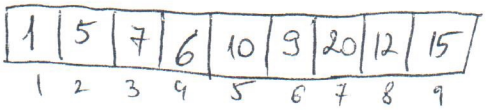
current



Swap with pos. 2 (elem (1))



swap 10 with 5 $\Rightarrow$



DONE.

b) In the worst case:

At level 0: there are 2^H nodes, where $H = \lfloor \log N \rfloor$, but cost = 0 (nothing to do)

level 1: —"— 2^{H-1} nodes, $\Rightarrow$ cost $= 1 \cdot 2^{H-1}$ for all of these nodes

⋮

level H-1: —"— 2^1 nodes, cost $= (H-1) \cdot 2^1$

level H (root): —"— 2^0 nodes, cost $= H \cdot 2^0$

(The cost of one swap + propagation of the swapped downwards)

$$\Rightarrow \sum_{h=0}^{H} h \cdot 2^{H-h} = \sum_{h=0}^{H} h \cdot \frac{2^H}{2^h} = 2^H \sum_{h=0}^{H} \frac{h}{2^h}$$

H = height of the heap $= \lfloor \log N \rfloor \Rightarrow 2^H = 2^{\lfloor \log N \rfloor} \leq N$

$$\Rightarrow 2^H \sum_{h=0}^{H} \frac{h}{2^h} \leq N \sum_{h=0}^{\lfloor \log N \rfloor} \frac{h}{2^h} \leq N \sum_{h=0}^{\infty} \frac{h}{2^h} = 2N = \boxed{O(N)}$$

$$\sum_{k=0}^{\infty} \frac{k}{2^k} = 2$$

$$\sum_{k=0}^{\infty} \frac{k}{2^k} = \frac{1}{2} + \frac{2}{2^2} + \frac{3}{2^3} + \dots$$

$$\sum_{k=0}^{\infty} \frac{k}{2^k} = \sum_{k=0}^{\infty} k x^k, \quad x = \frac{1}{2} \rightarrow \frac{x}{(1-x)^2} \rightarrow \frac{\frac{1}{2}}{(1-\frac{1}{2})^2} = 2$$

$$\sum_{k=1}^{N} k x^k = 1 \cdot x^1 + 2 \cdot x^2 + \dots + N \cdot x^N$$

$$= x(1 + 2x + 3x^2 + \dots + N x^{N-1})$$

$$= x(x + x^2 + x^3 + \dots + x^N)'$$

$$= x \cdot \left(\frac{x^N - 1}{x - 1}\right)' = x \cdot \frac{(N \cdot x^{N-1})(x-1) - (x^N - 1)}{(x-1)^2}$$

$$= x \cdot \frac{N \cdot x^N - N \cdot x^{N-1} - x^N + 1}{(x-1)^2} = \frac{(N-1)x^{N+1} - N \cdot x^{N-1} + 1}{(x-1)^2}$$

In the best case, the cost is ~~O(1)~~ no cost if the array is already arranged.

d) Combining two min heaps of length N.

If we consider the first heap and add elements to it, from the 2nd heap, one by one => complexity is $O(N \log N)$, where $\log N$ = cost of adding one element & maintaing the heap property.

If we merge the two arrays into one (concatenation), this operation will take $O(N)$, for adding each of the elements of the second heap to the buttom of the first heap. In order to arrange the array to represent a combined heap, we can apply the algorithm I described in point b), which has $\Theta(N)$.

=> Total cost is $\Theta(2N) = \Theta(N)$ ✓

d) $T(n) = 2\,T(\frac{n}{4}) + n^{0.5}$

$a = 2$, $b = 4$, $f(n) = \sqrt{n} = n^{0.5}$

$n^{\log_b a} = n^{\log_4 2} = n^{0.5} \Rightarrow f(n) = \Theta(n^{\log_4 2})$ (case 2)

$\Rightarrow T(n) = \Theta(n^{0.5} \log n)$

$T(n) = \Theta(\sqrt{n} \log n)$

e) $T(n) = 3\,T(\frac{n}{2}) + n \log n$

$a = 3$, $b = 2$, $f(n) = n \log n$

$\log_b a = \log_2 3 = 1.584$

$n \log n = O(n^{1.584-\varepsilon})$, $\varepsilon > 0 \Rightarrow T(n) = \Theta(n^{\log_2 3})$ (case 1)

from:

$n^{1.584} \neq n\sqrt{n}$

$n \log n < n\sqrt{n} \leq n^{1.584}$

$\log n < \sqrt{n}$

$\log 1 < \sqrt{1}$

$\log 2 < \sqrt{2}$

Suppose $\log n < \sqrt{n}$. Then $\log(2n) < \sqrt{2n}$

$\Rightarrow \log(2n) = \log 2 + \log n < \log 2 + \sqrt{n} < \sqrt{2n}$

$(\log 2)^2 + n + 2\sqrt{n} \log 2 < 2n$ |^2

$(\log 2)^2 + 2\sqrt{n} \log 2 < n$

True for $n > n_0$. $\in \mathbb{N}$.

Georgiona Tache
gxt6286

02/15/2014

Homework #2

CSCE 500

② a) $T(n) = 2T\left(\frac{n}{2}\right) + n^3$

$T(n) = a\,T\left(\frac{n}{b}\right) + c\,f(n)$

$a = 2$ $\quad f(n) = n^3$
$b = 2$

$n^{\log_b a} = n^{\log_2 2} = n \Rightarrow f(n) = \Omega\left(n^{1+\varepsilon}\right)$ (case 3)

and $a\,f\left(\frac{n}{b}\right) \le c\,f(n)$

$\Leftrightarrow \exists\, c < 1$ pos. such that $2\,f\left(\frac{n}{2}\right) \le c \cdot n^3$

$2 \cdot \frac{n^3}{4 \cdot 8} \le c\,n^3 \rightarrow c = \frac{1}{3}$ for $n > 0$

$\Rightarrow T(n) = \Theta(f(n)) = \Theta(n^3)$

b) $T(n) = 16\,T\left(\frac{n}{4}\right) + n^2$

$a = 16$ $\quad f(n) = n^2$
$b = 4$

$\log_b a = \log_4 16 = 2 \Rightarrow f(n) = \Theta(n^2)$ (case 2)

$\Rightarrow T(n) = \Theta(n^2 \log n)$

c) $T(n) = 7\,T\left(\frac{n}{2}\right) + n^2$

$a = 7$ $\quad f(n) = n^2$
$b = 2$

$2 < \log_b a = \log_2 7 < \log_2 8 = 3 \Rightarrow f(n) = O\left(n^{\log_2 7 - \varepsilon}\right)$ (case

$\exists\, \varepsilon,\ \varepsilon = 0.7$ for example $\Rightarrow T(n) = \Theta\left(n^{\log_2 7}\right)$

($\log_2 7 \approx 2.8$)