Assignment 2 Answers

COT 6405 - Introduction to Theory of Algorithms

1) **(4 points each question) Use the Master Theory to solve the following recurrences**

a. $T(n) = 3T(n/27) + 1$

$$a = 3, b = 27, f(n) = 1$$
$$n^{lg_b a} = n^{lg_{27} 3} = n^{1/3}$$

Now Compare with $f(n)$ $f(n) \in O(n^{1/3-\epsilon})$, case 1 applies:

$$T(n) = \theta(n^{lg_b a}) = \theta(n^{1/3})$$

b. $T(n) = 7T(n/8) + lgn$

$$a = 7, b = 8, f(n) = lgn$$
$$n^{lg_b a} = n^{lg_8 7} = n^{0.936}$$

Compare lgn and $n^{0.936}$, $lgn \in O(n^{0.936-\epsilon})$ when $\epsilon < 0.9$ case 1 applies:

$$T(n) = \theta(n^{lg_b a}) = \theta(n^{lg_8 7})$$

c. $T(n) = 2T(n/4) + n$

$$a = 2, b = 4, f(n) = n$$
$$n^{lg_b a} = n^{lg_4 2} = n^{1/2}$$

Now Compare with $f(n) = n$
$f(n) \in \Omega(n^{1/2+\epsilon})$,when $\epsilon <= 1/2$, AND for regularity condition:
$af(n/b) <= cf(n)$
$2(n/4) = n/2 < cn$ for $1/2 < c < 1$. Thus case 3 applies:

$$T(n) = \theta(n)$$

d. $T(n) = 2T(n/4) + n^2$

$$a = 2, b = 4, f(n) = n^2$$
$$n^{lg_b a} = n^{lg_4 2} = n^{1/2}$$

Compare with $f(n) = n^2$
$f(n) \in \Omega(n^{1/2+\epsilon})$, when $\epsilon <= 3/2$. AND for regularity condition:
$af(n/b) <= cf(n)$
$2(n/4)^2 = n^2/8 < cn^2$ for $1/8 < c < 1$. Thus Case 3 applies:

$$T(n) = \theta(n^2)$$

e. $T(n) = 2T(n/4) + \sqrt{n}lgn$

$$a = 2, b = 4, f(n) = \sqrt{n}lgn$$
$$n^{lg_b a} = n^{lg_4 2} = n^{1/2}$$

compare with $f(n) = \sqrt{n}lgn$
We see $f(n) = \theta(n^{lg_4 2}lg^k n)$, k = 1.
Case 2 applies:

$$T(n) = \theta(n^{lg_4 2}lg^{k+1}n) = \theta(\sqrt{n}lg^2 n)$$

2) **(10 points) Illustrate the operation of MAX-HEAPIFY (A, 1)on the array A = {27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0}.**

Answer: We can show the array as a heap here:

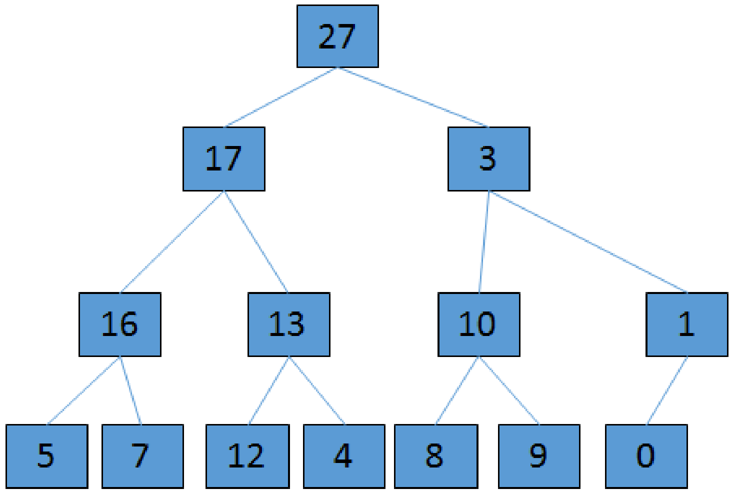


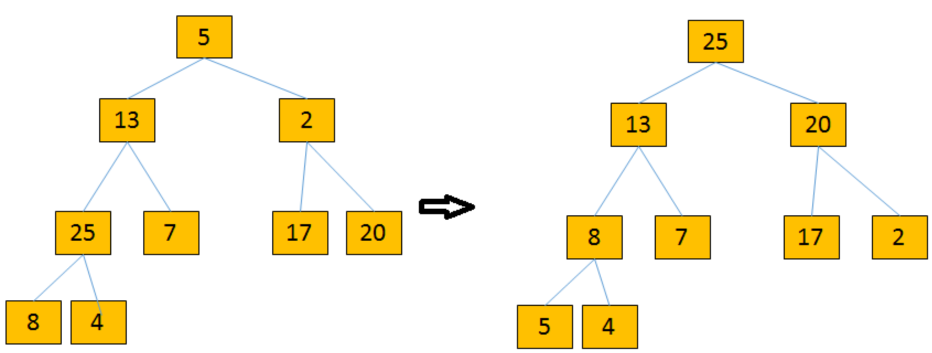
Figure 1: Heap Graph

Since A[1] is greater than A[2] and A[3], no swap is performed. In addition, the recursive function in swap condition is not called. So there is no actions.

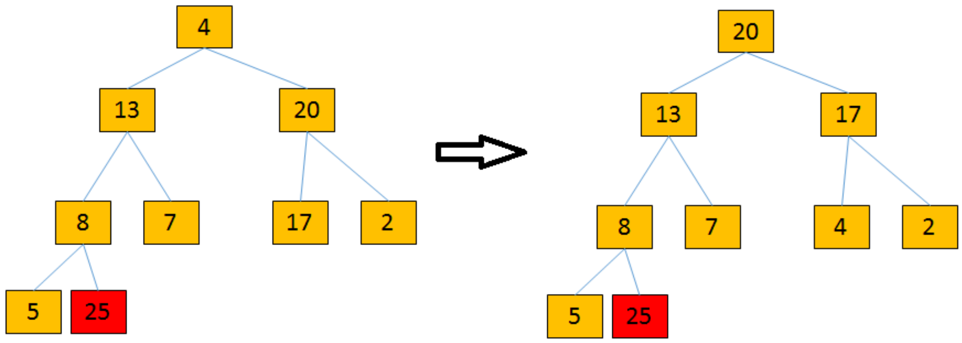
3) (10 points) (Textbook 6.4-1 page 160) Illustrate the operation of HEAPSORT on the array A ={5, 13, 2, 25, 7, 17, 20, 8, 4}.

Answer: First we show the heap graph and then we perform build-max-heap function.

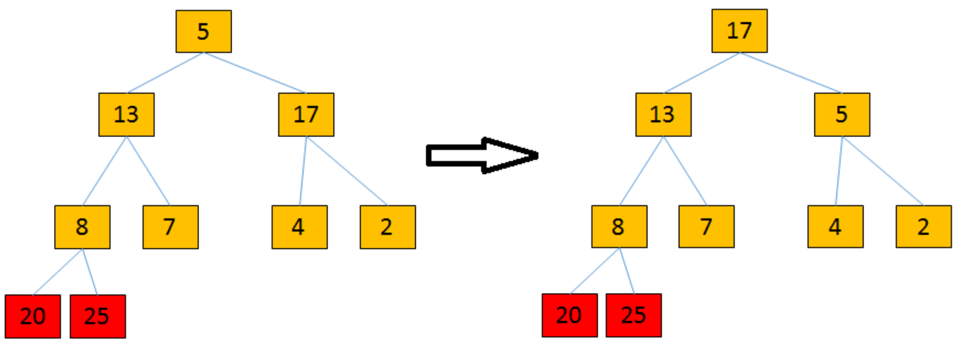
We start from A[4], 25 is greater than left and right childern. so no swap. 2 is swapped with 20. 13 is swapped with 25. Then 25 is swapped with 5. 5 is swapped with 13. After that 5 is swapped with 8. And the max heap is build as here.



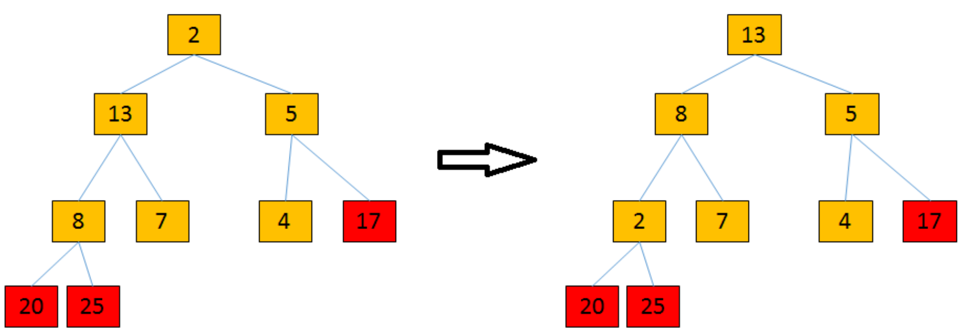
Next, we swap 25 with 4, call max-heapify for element 1. 4 and 20 is swapped, then 4 and 17 is swapped.



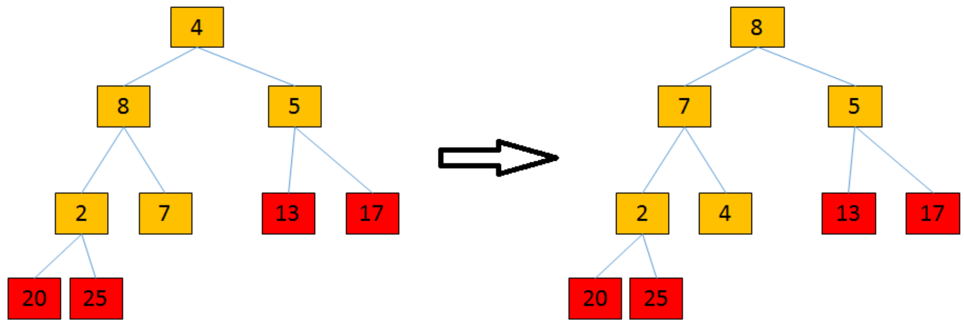
Next, we swap 20 with 5. call max-heapify. 5 is swapped with 17.



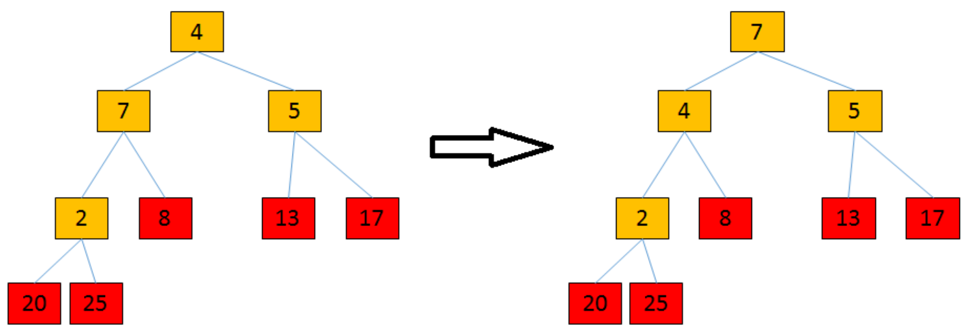
Next, we swap 2 with 17. call max-heapify. 2 is swapped with 13. Then 2 is swapped with 8.



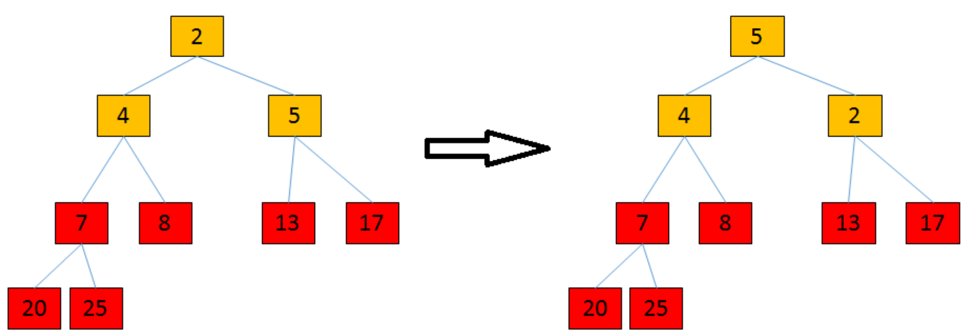
Next, we swap 13 with 4. call max-heapify. 4 is swapped with 8. Then 4 is swapped with 7.



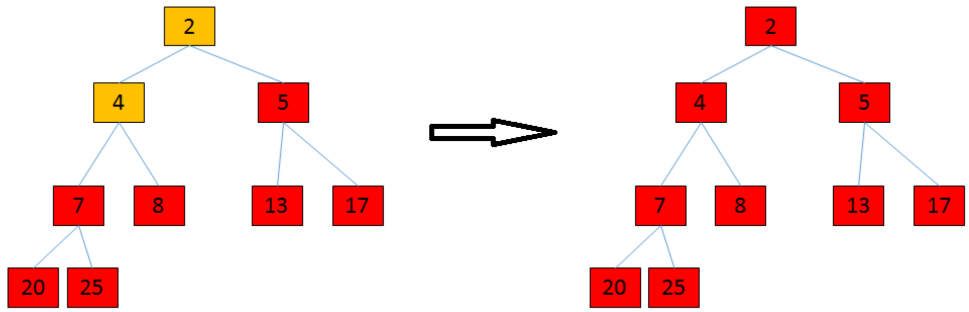
Next, we swap 8 with 4. call max-heapify. Then 4 is swapped with 7.



Next, we swap 7 with 2. call max-heapify. Then 2 is swapped with 5.



Swap 2 with 5. call max-heapify for the last 2 elements and swap last 2. The sort is done.



4) **(10 points) Use the substitution method to prove that $\mathbf{T}(\boldsymbol{n}) \in \Omega\,(nlgn)$ for the recurrence $\mathbf{T}(\boldsymbol{n}) = \mathbf{2T(0.5}\boldsymbol{n}\ \mathbf{-\ 3)} + \mathbf{n}$. In your proof, please do not simply ignore the constant to assume that $\mathbf{T(0.5}\boldsymbol{n}\ \mathbf{-\ 3)}$ is approximately equal to $\mathbf{T(0.5}\boldsymbol{n}\mathbf{)}$.**

Solution: For the lower bound, we guess the solution is $\mathrm{T}(n) = \Omega\ (n\mathrm{lgn})$. This means that we can find a constant d larger than 0 such that $\mathrm{T}(n) \geq d\ nlgn$

Assume: $\mathrm{T}(k) \geq d\ klgk$ for all $k \leq n$

Then, $\mathrm{T(n)} = 2\mathrm{T}(0.5n\text{ - }3) + \mathrm{n} \geq 2\mathrm{d}(0.5n\text{ - }3)\mathrm{lg}(0.5n\text{ - }3) + n$

$\geq 2\mathrm{d}(0.5n\text{ - }3)(\mathrm{lg}(0.5n)\text{ - }3) + n$

(Because $\mathrm{lg}(0.5n) - 3 = \mathrm{lg}(\ \frac{n}{16}\ ) \leq \mathrm{lg}(0.5n\text{ - }3) = \mathrm{lg}(\ \frac{n-6}{2}\ )$ when $\mathrm{n} \geq 7$)

$\geq 2\mathrm{d}(0.5n\text{ - }3)(\mathrm{lg}(0.5n)\text{ - }3) + n = (dn - 6d)\ (\mathrm{lg}(n)\text{ - }4) + n$

$= dn\mathrm{lgn} - 4dn\text{ - }6dlgn + 24d + n \geq dn\mathrm{lgn} - 4dn\text{ - }6dlgn + n$ (Because $24d > 0$)

$\geq dn\mathrm{lgn} - 4dn\text{ - }6dn + n$ (because $\mathrm{lg}n<n$)

$\geq\ \ dn\mathrm{lg}n$ (when $-4dn\text{ - }6dn + n \geq 0, that\ is\ d \leq 0.1$)

For the upper bound, $\mathrm{T}(n) = 2\mathrm{T}(0.5n\text{ - }3) + n \leq 2\mathrm{T}(0.5n) + n \in O\,(nlgn)$, therefore $\mathrm{T}(n) \in \Theta\,(nlgn)$.

5) For HEAPSORT codes below

```
Heapsort(A)
{
        Build-MAX-Heap(A);
        for (i = A.length down to 2)
```

```
        {
            Swap(A[1], A[i]);
            A.heap_size= A.heap_size - 1;
            MAX-Heapify(A, 1);
        }
}
```

a) (3 points) What is the number of required swap operations when heapsort the array A = { 5, 13, 2, 25, 7, 17, 20, 8, 4} ? Explain your reason.

Answer:
23 total swaps.
5 initial swaps to Build-MAX-Heap.
8 Heapsort swaps to exchange the root with the last leaf.
10 MAX-Heapify swaps during Heapsorting.

b) (3 points) If we replace MAX-Heapify(A, 1) with Build-MAX-Heap(A), what is the number of required swap operations when heapsort the array A? Explain your reason.

Answer:
There will be the same number of swaps. Each Heapsort root/last leaf swap ensures that the root of the heap is not a max heap, but each subtree is a max heap from previous MAX-Heapify calls. Therefore, Build-MAX-Heap will simply be searching until the root for a heap in can MAX-Heapify.

c) (4 points) Does the asymptotic upper bound of Heapsort increase from O(nlgn) to O(n^2)? Why? (Hint: compare the number of swap operations before and after the change for the worst case).

Answer:
For the worst case:

MAX-Heapify is $O(lgn)$.
Build-MAX-Heap is $O(nlgn)$

Heapsort using MAX-Heapify is
$O(nlgn) + (n-1)O(lgn) = O(nlgn) + O(nlgn) = O(nlgn)$

Heapsort with MAX-Heapify replaced with Build-MAX-Heap is
$O(nlgn) + (n-1)O(nlgn) = O(nlgn) + O(n^2lgn) = O(n^2lgn) = O(n^2)$

6) **(10 points) Can we use the Master Theory on the recurrence $\mathbf{T(n) = 2T(n/2) + \sin(n)}$? Please answer YES or NO and then explain your reason. Can we use the Master Theory on the recurrence $\mathbf{T(n) = T(n/2) + n\sin(n) + 2n}$? Please answer YES or NO and then explain your reason.**

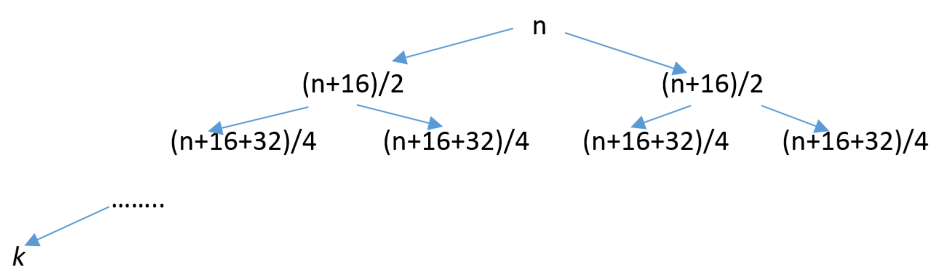
Solution: No for the first recurrence, because sin(n) is not asymptotically positive.

No for the second recurrence, if we use master theory, $n^{log_b a} = 1$, and $f(n) = n\sin(n) +2n \in \Omega(1)$. It seems that this fits case 3 of master theory. However, when we check the regularity condition $af(n/b) \leq cf(n)$, we find that

$af(n/b) = \frac{n}{2}\sin(n/2) +n$ and both $\frac{n}{2}\sin(n/2) +n and$ $n\sin(n) +2n$ are periodic functions. It is impossible to find a constant c <1 such that $\frac{n}{2}\sin(n/2) +n$ is always smaller than $n\sin(n) +2n$

7) **(10 points) Use the recursion tree method to determine the asymptotic upper and lower bounds for the recurrence T(n) = 2T ($\frac{n}{2}$ + 8) + n.**

Answer:



The subproblem size for a node at depth i is $[n + 8(2^1 + 2^2 + ..2^i)]/2^i$ = (n + $2^{i+4} - 16$)/ 2^i ($i \geq$ 1) and the subproblem size for the root node ($i = 0$) is n. Assume the subproblem "bottoms out" when its size hits k, where k is an integer constant. This means (n + 2^{i+4} – 16)/ 2^i = k and i = lg($\frac{16-n}{16-k}$).

So the tree has 1+ lg($\frac{16-n}{16-k}$) levels

- Each node at level i has a cost of (n + 2^{i+4} – 16)/ 2^i
- Each level has 2^i nodes
- Thus, the total cost of level i is 2^i (n + 2^{i+4} – 16)/ 2^i = n + 2^{i+4} – 16 when $i \geq$ 1, and when $i < 1$, the cost for the root node is n
- The bottom level has 2^i nodes, each costing T(1)
- Assume T(1) is a constant. The total cost of the bottom level is Θ (2^i)
- The total cost for the entire tree is

- T(n) = n + $\sum_{i=1}^{lg\left(\frac{16-n}{16-k}\right)-1} (n + 2^{i+4} - 16)$ + $2^{lg\left(\frac{16-n}{16-k}\right)}$

$= \text{n} + \text{(n-16) lg(} \frac{16-n}{16-k} \text{)} + 16\text{(} \frac{1-2^{lg\left(\frac{16-n}{16-k}\right)}}{1-2} - 1 \text{)} + \frac{16-n}{16-k}$

$= \text{n} + \text{(n-16) lg(} \frac{16-n}{16-k} \text{)} + 16\text{(} \frac{16-n}{16-k} - 2 \text{)} + \frac{16-n}{16-k}$

$= \text{n} + \text{(n-16) lg(} \frac{n-16}{k-16} \text{)} + 16\text{(} \frac{n-16}{k-16} - 2 \text{)} + \frac{n-16}{k-16}$

$= \text{n} + \text{(n-16) lg(n-16)} - \text{(n-16)lg(k-16)} + 16\text{(} \frac{n-16}{k-16} - 2 \text{)} + \frac{n-16}{k-16}$

$= \text{n} + \text{nlg(n-16)- 16lg(n-16)} - \text{(n-16)lg(k-16)} + 16\text{(} \frac{n-16}{k-16} - 2 \text{)} + \frac{n-16}{k-16}$

Let f(n) = n -16lg(n-16) − (n-16)lg(k-16) + 16($\frac{n-16}{k-16} - 2$) + $\frac{n-16}{k-16}$. Note that $lim_{n\to\infty}\frac{f(n)}{nlg(n-16)} = 0$. Therefore f(n) = o(nlg(n-16)), and we have

T(n) = nlg(n-16) + o(nlg(n-16)) = Θ(nlg(n-16)) = $\Theta\,(nlgn)$

8) **(10 points) Use mathematical induction to prove the correctness of the Build-MAX-Heap function.**
Answer:

Worst base case:

A = {1, 2, 3}, P(3)

Build-MAX-Heap starts at the root: A.length / 2 = 3 / 2 = 1.5 = 1

It swaps 3 with 1. We then have a max heap: A = {3, 2, 1}

Assume P(n) holds: A is a max heap of n elements, consisting of all of the elements of A.

P(n + 1): For the worst case, add an element e to the end of A (at A[n + 1]) that is greater than all elements in A.

Build-MAX-Heap starts at e's parent: A[n + 1 / 2]. Because it is greater than its parent and the other leaf, it will be swapped with its parent. Element e now resides at A[n + 1 / 2]. Both subtrees are max heaps.

Build-MAX-Heap will now decrement through subtrees which are already max heaps, until it reaches A[n + 1 / 4], the parent of e. It will be swapped. Element e now resides at A[n + 1 / 4]. Both subtrees are max heaps.

Repeat until e resides at A[2]. Build-MAX-Heap will consider A[1], the root tree. Element e will be greater than the root, so it will be swapped. Element e, the maximum element of A, now resides at A[1]. Both subtrees are max heaps. Thus, A is a max heap for the case P(n + 1).

Therefore, Build-MAX-Heap is correct.