[CS 251] Homework 1 Solution.

#1. $D(N)$: Divide-and-conquer recurrence

$\Rightarrow D(N) = 2D(\frac{N}{2}) + N \log N$, $D(1) = 0$.

Dividing it by N, we have

$$\frac{D(N)}{N} = \frac{D(\frac{N}{2})}{\frac{N}{2}} + \log N. \longleftarrow N = 2^n.$$

$$\frac{D(2^n)}{2^n} = \cancel{\frac{D(2^{n-1})}{2^{n-1}}} + n$$

$$\cancel{\frac{D(2^{n-1})}{2^{n-1}}} = \cancel{\frac{D(2^{n-2})}{2^{n-2}}} + n-1$$

$$\vdots$$

$$+ \big) \; \cancel{\frac{D(2)}{2}} = \cancel{D(1)}^{\,0} + 1$$

$$\frac{D(2^n)}{2^n} = n + (n-1) + \cdots + 1 = \frac{1}{2}n(n+1)$$

$$\therefore D(2^n) = 2^{n-1} n(n+1)$$

$$\Rightarrow D(N) = \frac{1}{2}N \log N(\log N + 1) = O(N(\log N)^2).$$

$\boxed{\text{Ans : 4}}$

#2. $D(N) = 2D(\frac{N}{2}) + N\sqrt{N}$, $D(1) = 0$.

$$\Rightarrow \frac{D(N)}{N} = \frac{D(\frac{N}{2})}{\frac{N}{2}} + \sqrt{N} \longleftarrow N = 2^n.$$

$$\frac{D(2^n)}{2^n} = \cancel{\frac{D(2^{n-1})}{2^{n-1}}} + (\sqrt{2})^n$$

$$\cancel{\frac{D(2^{n-1})}{2^{n-1}}} = \cancel{\frac{D(2^{n-2})}{2^{n-2}}} + (\sqrt{2})^{n-1}$$

$$\vdots$$

$$+ \big) \; \cancel{\frac{D(2)}{2}} = \cancel{D(1)}^{\,0} + \sqrt{2}$$

$$\frac{D(2^n)}{2^n} = \sqrt{2} + \cdots + (\sqrt{2})^n = \frac{\sqrt{2}(\sqrt{2}^n - 1)}{\sqrt{2} - 1}$$

$$\therefore D(2^n) = \frac{\sqrt{2}}{\sqrt{2}-1} \cdot 2^n \cdot (\sqrt{2}^n - 1)$$

$$D(N) = \frac{\sqrt{2}}{\sqrt{2}-1} \cdot N(\sqrt{N} - 1) = O(N\sqrt{N}).$$

$\boxed{\text{Ans : 1}}$

#3. Keep in mind : "log" grows much slower than the linear functions !

① $(\log\log n)^2 \quad (\log n)^{0.7}$

$\Updownarrow \log n = x$

$(\log x)^2 \quad x^{0.7}$

$\Updownarrow$ take log

$2\log\log x \quad 0.7\log x$

$\Updownarrow \log x = t$

$2\log t \quad 0.7t$

$\Updownarrow$

$\log t \quad 0.35t$

$\Rightarrow \log t < 0.35t$, therefore $(\log\log n)^2 < (\log n)^{0.7}$ for large n.
(for large t)

② $(\log n)^{0.7} \quad \sqrt{n}$

$\Updownarrow$ take log

$0.7\log\log n \quad 0.5\log n$

$\Updownarrow \log n = x$

$\log x \quad \frac{5}{7}x$ Similar to ①, $(\log n)^{0.7} < \sqrt{n}$ for large n.

③ (Stirling's formula) $\log(n!) \sim n\log n$

$\Rightarrow \sqrt{n} < n < n\log n \sim \log(n!)$ for large n.

④ $\log(n!) \sim n\log n \quad n^{1.1}$

$\Updownarrow$

$\log n \quad n^{0.1}$

$\Updownarrow$ take log

$\log(\log n) \quad 0.1\log n$

$\Updownarrow \log n = x$

$\log x \quad 0.1x$ Similar to ①, $\log(n!) < n^{1.1}$ for large n.

⑤ Trivially, $n^{1.1} < n^2 < \underbrace{2^n < n!}_{2^n < n^n \sim n!}$ for large n.

Stirling

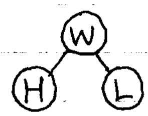
$\boxed{\text{Ans : 4}}$

#4.

insert H

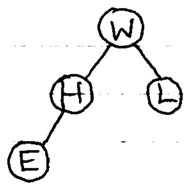
insert W

insert L



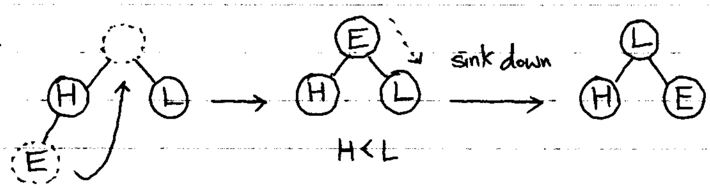
W > L, no swim.

insert E

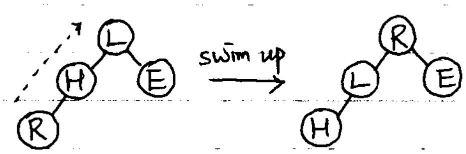


H > E, no swim.

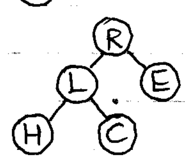
remove max



insert R

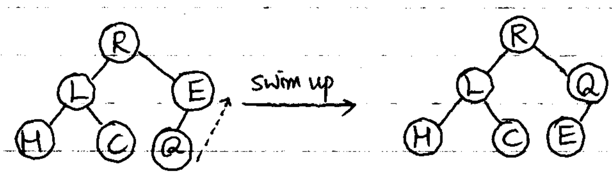


insert C

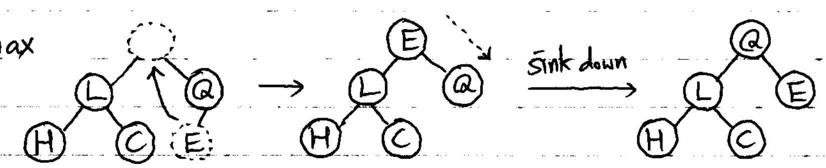


L > C, no swim.

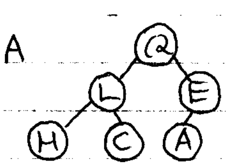
insert Q



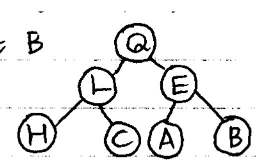
remove max



insert A



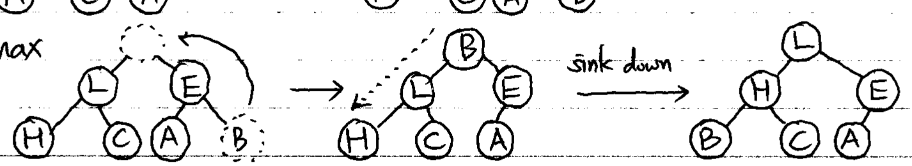
insert B



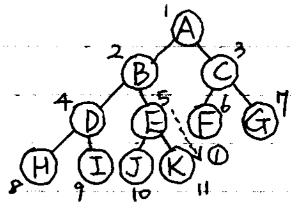
L H E B C A.

$\boxed{\text{Ans : 4}}$

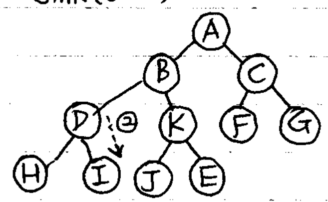
remove max



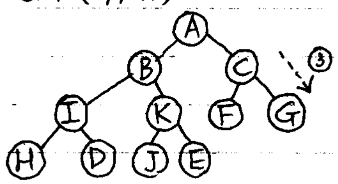
#5. A B C D E F G H I J K



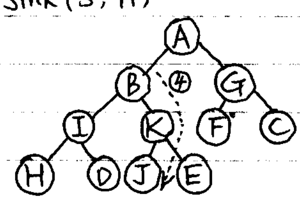
① sink(5, 11)



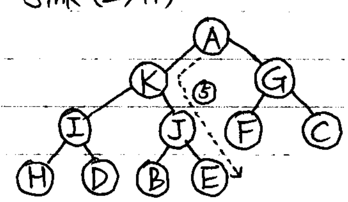
② sink(4, 11)



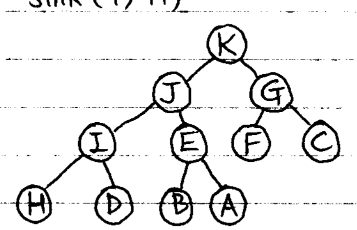
③ sink(3, 11)



④ sink(2, 11)



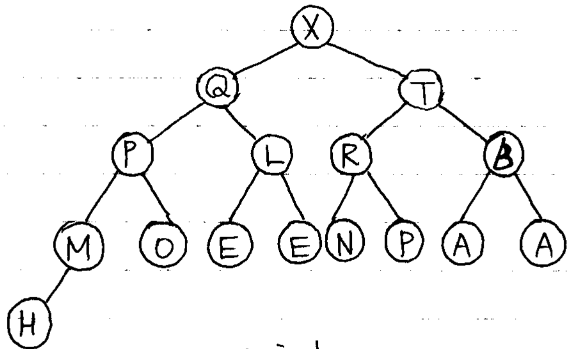
⑤ sink(1, 11)



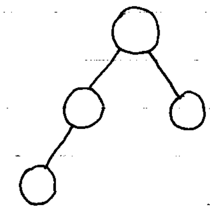
⇒ K J G I E F C H D B A.

$\boxed{\text{Ans : 2}}$

#6. Let's look at the example for the case $N = 2^4 = 16$ and $n = 4$.

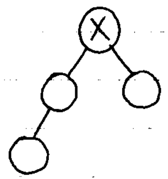


main heap



auxiliary heap

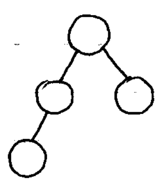
- Insert, in the auxiliary heap, the key at the root of the main heap.



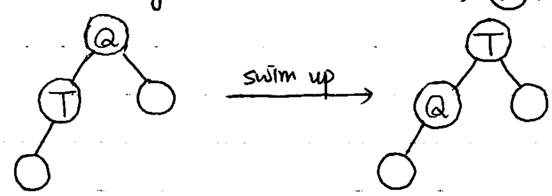
Note : $n-1 = 3$.

- [1st repetition]

  - Remove-max ⇝ X

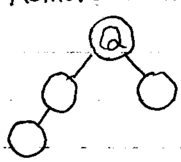


  - Insert, the keys of the two children of X in the main heap

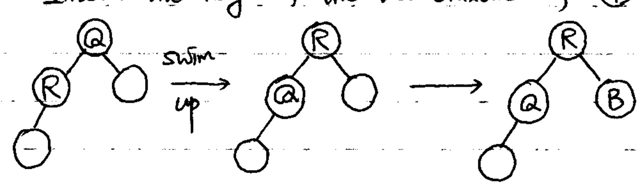


[2nd repetition]

- Remove-max ⇝ T

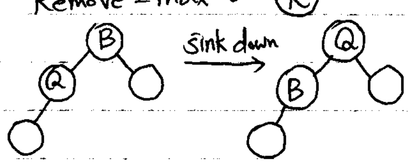


- Insert, the keys of the two children of T in the main heap

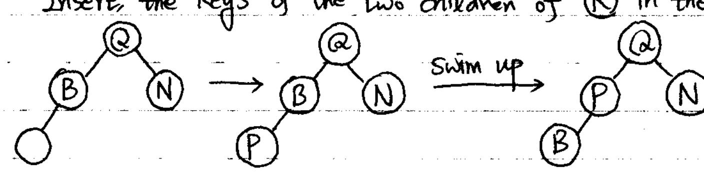


[3rd repetition]

- Remove-max ⇝ R



- Insert, the keys of the two children of R in the main heap

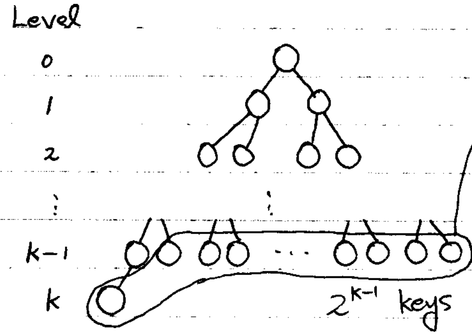


- Return the maximum element in the auxiliary heap. → Q

Here, Q is the 4th largest key in the main heap.

$\boxed{\text{Ans} : 4}$

#7. Assume that $n = 2^k$.



If one of these keys are exchanged with root when removing max, we expect $2(k-1)$ compares to sink down.

$\Downarrow$

Time complexity of total remove-max operations in the algorithm is

$$\sum_{t=0}^{k-1} 2t \times 2^t$$

Let this value be s. Then

$$s = 2^2 + 2\cdot 2^3 + 3\cdot 2^4 + \cdots + (k-1)\cdot 2^k \quad \cdots ①$$

$$2s = 2^3 + 2\cdot 2^4 + \cdots + (k-2)\,2^k + (k-1)\cdot 2^{k+1} \quad \cdots ②$$

② − ① gives

$$s = -(2^2 + 2^3 + 2^4 + \cdots + 2^k) + (k-1)\,2^{k+1}$$

$$= (k-2)\,2^{k+1} - 4 = O(k\,2^k) = O(n \log n).$$

Similarly, we have the time complexity of total insert operations in the algorithm as $O(n \log n)$. Therefore, total time is $O(n\log n) + O(n \log n) = O(n\log n)$.

$\boxed{\text{Ans : 3}}$

#8 ~ #10. Intuitively, $L[i] = \sum_{k=0}^{i} A[k]$, $M[i] = \min_{0\le k\le i} L[k]$, $R = \max_{0\le k\le N-1} D[k]$

returns the largest $S[i,j]$. Note that $A[0]=0$ to make this algorithm works to properly compare the partial sums at each step.

$\boxed{\text{Ans : 1, 2, 3}}$

ex)				
A[i]	a=0	b	c	d
L[i]	0	b	b+c	b+c+d
M[i]	0	0 or b	0 or b or b+c	0 or b or b+c or b+c+d
D[i]=L[i]−M[i]	0	0 or b	0 or c or b+c	0 or d or c+d or b+c+d

We covered all possible cases