Homework 2

Team: nogg[*][†]

October 24, 2016

Exercise 1.

Solution.
Observation.1 We can find that the array A's size is n and n is even, so we can divide the array into $\frac{n}{2}$ pairs.
Observation.2 If we have two arrays, the biggest number among these arrays must be either the first array's biggest number or the second array's biggest number.

step1. Divide the array into $\frac{n}{2}$ pairs which counts from 1 to $\frac{n}{2}$.
step2. Inside each pairs we do one comparison to determine which one is bigger and which one is smaller.
step3. Now we get two arrays, one is composed of the bigger one of each pair, another is composed of the smaller one of each pair.
step4. Find the biggest number N_b in the first array and the smallest number N_s in the second array.
step5. Now the biggest number of A is N_b and smallest number of A is N_s.

Analysis: In step2 we do $\frac{n}{2}$ comparisons, and in step4 we do $(\frac{n}{2}$-1)+($\frac{n}{2}$-1).Totally, we do $\frac{n}{2}$ + ($\frac{n}{2}$-1) + ($\frac{n}{2}$-1) = $\frac{3n}{2}$-2 comparisons. □

Exercise 2.

Solution. **Observation.** We have known that we need at least (n-1) times comparisons to find the biggest numbers.

step1. We find the biggest number in the array through Dividing and Comparing. That is to say, assuming we have a full binary tree which has 2^k leaf nodes, and every leaf store a number of the array, and the inside node stores a value which is the bigger one of his sons.

[*]E-mail: kimi.ysma@gmail.com
[†]Team member: Ma Yesheng, Zhao Ming, Hu Hu, Zou Yikai, Fan Minghua

So we have to do $(2^{k-1} + 2^{k-2} + ... + 1) = (2^k - 1)$ comparisons to find the biggest one. i.e (n-1) comparisons.

Observation. The second-biggest number in the array must have done comparison with the biggest one during the construction of the full binary tree.
Proving. Assuming the biggest number is N_1, and the second-biggest number is N_2. If N_2 hasn't compared with N_1, it must be smaller that a number, and that number is not N_1, so N_2 is at least third-biggest number, which is not same with the assumption.

step2. Get the numbers which have compare with N_1, there will be $log_2(n)$, which is the height of the full binary tree. Find the biggest number of these $log_2(n)$ numbers.

Analysis: In step1. we do (n-1) comparisons, and in step2. we do ($log_2(n)$-1) comparisons. Totally, we do (n + $log_2(n)$ - 2) comparisons. □

Exercise 3.

Solution. **Analysis:** The algorithms say that the maximum compare time is n-2.
There are n numbers, and the only way to access the elements is to take them and compare them pairwise.So for each comparison, no matter which two numbers we choose to compare, we can only eliminate one number that is not the larger number we want. So after n-2 comparisons, there will still be two numbers left, and we wont know anything about the remaining two numbers so that we cant find the maximum number in the array.
So the algorithm will certainly fail on certain inputs. □

Exercise 4.

Solution.
Analysis: Since we want to get the worst case, which means at every merge level, when one list runs out for adding, the other must have only one element left. Then we can construct the list in this way, which I prefer to call it **Interpolation**. Now consider the last merge step's reversion: dividing one list(lenth is n) into two lists(lenth is n/2). The interpolation method is to divide it crossways. Assuming the origin list is range(n), then the two new lists are like: [0, 2, 4, ..., n-2] and [1, 3, 5, ...,n-1].Then let's move on, dividing these two lists into four using the same method.Now we can get[0, 4, ..., n-4], [2, 6, ..., n-2], [1, 5, ...,n-3] and [3, 7, ..., n-1].And so on until each list contains only one element, and now explain why it is actually the worst case. If we combine two lists with lenth(n) into one, the elements in previous lists are cross-growing, like a[0]<b[0]<a[1]<b[1]<....Then we need $2n - 1$ times of addtion to join them. Then we add all the comparion time up, we get $nlogn - n + 1$.
Observation: Let's denote that the origin list is at level 0, and the two lists are at level 1 and so on. Then we can get that: at level i, the span in each list is 2^i. And the leaves of this tree is at level log(n), which are lists with only one element.
Ok,I think what I explain above is quite enough but not explicit. Let's forget it, I just came

up a beautiful and tricky way to construct.
Process: If we regard every list(no matter how many elements in it) as a node, then the whole merge process is a **Full Binary Tree** ,whose leaves are one number and height is log(n)+1. Each node has a unique route from root to it. If at each inner node, the route is gonna choose the left way, then we denote it as 0, right as 1. Then each route has a unique **binary code**, which is obviously the index of the leaf node in binary. For example, in range(8), leaf no.0's route code is 000 which means left left left, and leaf no.5's route code is 101 which means right left right.In interpolation method, the list's min element will definitely appear in its left child, and max element in its right child. Then we can use this route code plus elements's span in each level to calculate the exact number of each leaf node.

(1) max=n-1, min=0, span=1, lenth=n, route code=(from 0...0$^{logn's}$ to 1...1$^{logn's}$), i=logn

(2) check if i is now 0, if it is, goto (4),else goto (3)

(3) if route code[i]=0, then min remains still, span = span*2, lenth = lenth/2, calculate max by $(max - min)/span + 1 = lenth$, i=i-1, else max remains still, span = span*2, lenth=lenth/2, calculate min by $(max - min)/span + 1 = lenth$, i=i-1. Goto (2)

(4) leaf[route code] = min(or max)

□

Exercise 5.

Solution. Let me introduce two methods to solve this problem:
Method 1:
Let us follow the step we use in class but do more precise work.(Denote that Aii=0 not 1)

$$\begin{aligned}
\sum_{i\neq j}\mathbb{E}[A_{ij}] &= \sum_{i\neq j}\frac{1}{|i-j|+1}\\
&= 2\sum_{i=1}^{n}\sum_{l=1}^{n-i+1}\frac{1}{l} - n\\
&= 2*(n*1+(n-1)*\frac{1}{2}+...+2*\frac{1}{n-1}+1*\frac{1}{n}) - n\\
&= 2*((n+1)H_n - n)) - 2n\\
&= 2(n+1)H_n - 4n
\end{aligned}$$

Method 2:
Let's define that E_n means expected comparisons needed when sorting list with length(n). And since we use kind of shuffle things to choose the pivot, then after first round comparison, the possibility that the pivot in each position is equal, in other words, $\frac{1}{n}$. And when it's at pos

i, then it divides the list into to parts with lengthi-1 and n-i. Then the total comparison can be represented like this $E_n = E_{i-1} + E_{n-i} + n - 1$. In consequence, the expected comparisons is actually $E_n = \frac{1}{n}\sum_{i=1}^{n}(E_{i-1} + E_{n-i} + n - 1)$. Now let's do some mathematic things.

$$\begin{aligned} n * E_n &= 2(E_1 + E_2 + ... + E_{n-1}) + n(n-1) && (1)\\ (n-1) * E_{n-1} &= 2(E_1 + E_2 + ... + E_{n-2}) + (n-1)(n-2) && (2)\\ nE_n &= (n+1)E_{n-1} + 2n - 2 && ((1)\text{-}(2)) \end{aligned}$$

Now we can get a series of equations. And then by replacing E_i with $\frac{i+1}{i}E_{i-1} + 2 - \frac{2}{i}$ until we get:

$$\begin{aligned} E_n &= \frac{n+1}{2}E_1 + 2(1 + \frac{n+1}{n} + ... + \frac{n+1}{3}) - 2(\frac{n+1}{n(n+1)} + \frac{n+1}{n(n-1)} + ... + \frac{1}{3*2})\\ &= \frac{n+1}{2} * 0 + 2(n+1)(\frac{1}{3} + \frac{1}{4} + ... + \frac{1}{n} + \frac{1}{n+1} - \frac{1}{2} + \frac{1}{n+1})\\ &= 2(n+1)(H_n - 2 + \frac{2}{n+1})\\ &= 2(n+1)H_n - 4n \end{aligned}$$

□

Exercise 6.

Solution. In quicksort, we use a pivot number randomly chosen to divide the number sequence ito two part. Then we need to do the same quicksort operation on each part and repeat it. But in quickselect, similarly in each recursion step, we need to divide the sequence into two part, but we only need to do the same quickselect operation on one of them. Which means that, in quickselect, we only need to do "partial execution" in each step of quicksort. Therefore, quickselectcan be viewed as a "partial execution" of quicksort.

We use the sequence {4, 7, 9, 3, 1, 2, 6, 5, 8} as an example. The quicksort tree is drawn above.
In order to select the $5th$ number in this sequence, the part in red circle is visited by quickselect.

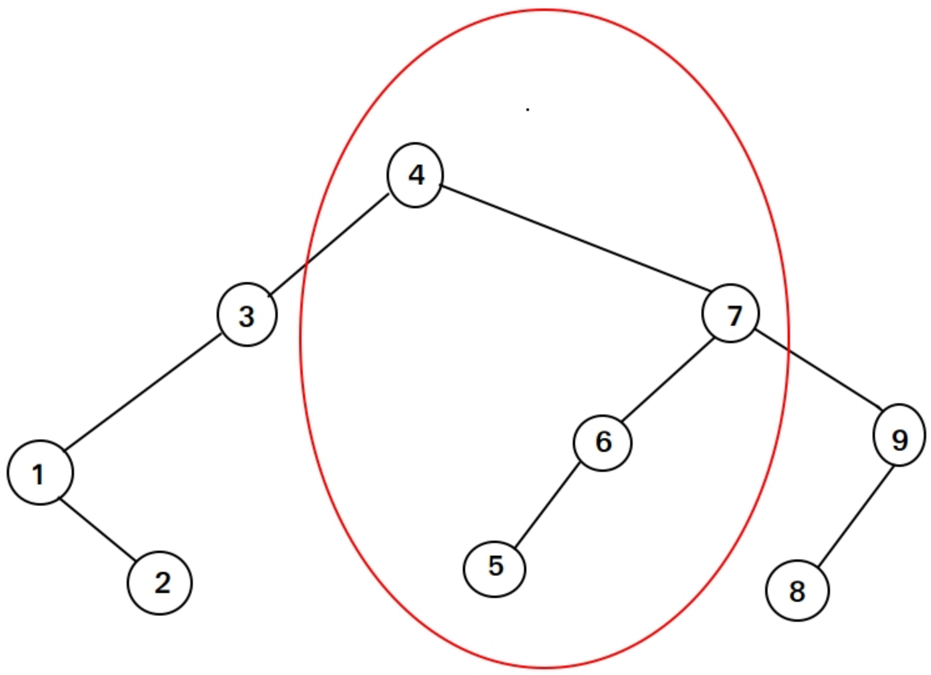

Figure 1: quicksort tree of [4, 7, 9, 3, 1, 2, 6, 5, 8]

□

Exercise 7.

Solution.

We want i to be the common ancestor of j and k. Consider a sequence which include i, j, k. If we want i be the ancestor of j and k, and assumed i < j < k, we must make sure that the pivot number we choose is j. It means that the pivot we choose must be the second big number in i, j, k.

$$\mathbb{E}[B_{i,j,k}] = \frac{1}{max\{i,j,k\} - min\{i,j,k\} + 1}$$

□

Exercise 8.

Solution. From analysis of the search route of SelectSort, we can find that the number of comparisons need to do in SelectSort can be counted by all the i, j, k where k is the argument of SelectSort, s.t. $B_{i,j,k} = 1$. Hence, we can define $C(\pi)$ to be:

$$C(\pi) = \sum_{\substack{i,j \\ i \neq j}} B_{i,j,k}$$

$\square$

Exercise 9.

Solution. With the equation of $C(\pi)$, we can go to compute the expectation of comparisons:

$$\begin{aligned}
\mathbb{E}[C(\pi)] &= \mathbb{E}\left[\sum_{i\neq j} B_{i,j,k}\right] = \sum_{i\neq j}\mathbb{E}[B_{i,j,k}] \\
&= 2\sum_{i>j}\mathbb{E}[B_{i,j,k}] = 2(\sum_{\substack{i>j\\1\leq k\leq j}}\mathbb{E}[B_{i,j,k}] + \sum_{\substack{i>j\\j\leq k\leq i}}\mathbb{E}[B_{i,j,k}] + \sum_{\substack{i>j\\i\leq k\leq n}}\mathbb{E}[B_{i,j,k}]) \\
&= 2(\sum_{\substack{i>j\\1\leq k\leq j}}\frac{1}{i-k+1} + \sum_{\substack{i>j\\j\leq k\leq i}}\frac{1}{i-j+1} + \sum_{\substack{i>j\\i\leq k\leq n}}\frac{1}{k-j+1}) \\
&= 2(\sum_{i=k+1}^{n}\sum_{j=k}^{i-1}\frac{1}{i-k+1} + \sum_{i=k+1}^{n}\sum_{j=1}^{k}\frac{1}{i-j+1} + \sum_{i=2}^{k}\sum_{j=1}^{i-1}\frac{1}{k-j+1}) \\
&= 2\left[(\sum_{i=2}^{n-k+1}\frac{i-1}{i}) + (\frac{(nH_n - H_n) - (kH_k - H_k)}{2} + H_n - H_k) + (\sum_{i=2}^{i=k}\frac{i-1}{i})\right]
\end{aligned}$$

(The second term of the above derivation is a little tricky, but if you consider a $n \times n$ matrix with each row a harmonic series, it is quite obvious)

$$\begin{aligned}
&= 2\left[(n-k+1-H_{n-k+1}) + (\frac{(n+1)H_n - (k+1)H_k}{2} + (k - H_k)\right] \\
&= 2(n+1) - 2H_{n-k+1} + (n+1)H_n - (k+3)H_k
\end{aligned}$$

Thus the expectation of comparison number is calculated as above. $\square$