CS 217 – Algorithm Design and Analysis

Shanghai Jiaotong University, Fall 2019

Handed out on Thursday, 2019-09-19
First submission and questions due on Thursday, 2019-09-26
You will receive feedback from the TA.
Final submission due on Monday, 2019-10-07

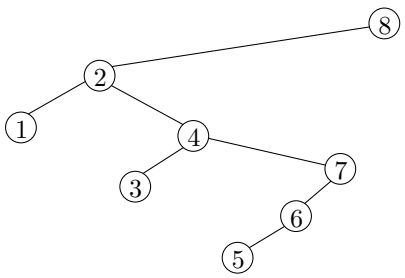
2 Sorting Algorithms

Exercise 1. Given an array A of n items (numbers), we can find the maximum with $n-1$ comparisons (this is simple). Show that this is optimal: that is, any algorithm that does $n-2$ or fewer comparisons will fail to find the maximum on some inputs.

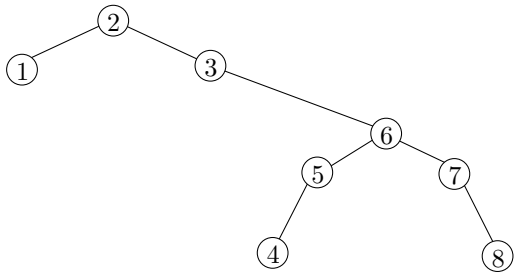
Exercise 2. Let A be an array of size n, where n is even. Describe how to find both the minimum and the maximum with at most $\frac{3}{2}n-2$ comparisons. Make sure your solution is *simple*, in describe it in a clear and succinct way!

Exercise 3. Given an array A of size $n = 2^k$, find the second largest element element with at most $n + \log_2(n)$ comparisons. Again, your solution should be *simple*, and you should explain it in a clear and succinct way!

Recall the quicksort tree defined in the lecture.



quicksort tree of $[8, 2, 4, 1, 7, 6, 5, 3]$



quicksort tree of $[2, 3, 6, 1, 7, 5, 8, 4]$

We denote a specific list (ordering) by π and the tree by $T(\pi)$. $A_{i,j}$ is an indicator variable which is 1 if i is an ancestor of j in the tree $T(\pi)$, and 0 otherwise. In the lecture, we have derived:

$$\mathbb{E}[A_{i,j}] = \frac{1}{|i-j|+1}$$
$$\text{total number of comparisons } = \sum_{i \neq j} A_{i,j} \ .$$

Exercise 4. Determine the expected number of comparisons made by quicksort. Your final formula must be *closed*, meaning it must not contain $\mathbb{E}$, $\prod$, or $\sum$. It may, however, contain $H_n := 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$, the n^{th} Harmonic number. **Remark.** This gets a bit tricky, and you will need some summation wizardry towards the end.

2.1 Quickselect

Remember the recursive algorithm QuickSelect from the lecture. I write it below in pseudocode. In analogy to quicksort we define QuickSelect deterministically and assume that the input array is random, or has been randomly shuffled before QuickSelect is called. We assume that A consists of distinct elements and $1 \leq k \leq |A|$.

Let C be the number of comparison made by QuickSelect. In the lecture we proved that $\mathbb{E}[C] \leq O(n)$ when we run it on a random input.

Exercise 5. Explain how QuickSelect can be viewed as a "partial execution" of quicksort with the random pivot selection rule. Draw an example quicksort tree and show which part of this tree is visited by QuickSelect.

Algorithm 1 Select the k^{th} smallest element from a list A

```
1: procedure QuickSelect(X, k)
2:     if |X| = 1 then
3:         return X[1]
4:     else:
5:         p := X[1]
6:         Y := [x ∈ X | x < p]
7:         Z := [x ∈ X | x > p]
8:         if |B| = k − 1 then
9:             return p
10:        else if |Y| ≥ k then
11:            return QuickSelect(Y, k)
12:        else
13:            Return QuickSelect(Z, k − |Y| − 1)
14:        end if
15:    end if
16: end procedure
```

Let $B_{i,j,k}$ be an indicator variable which is 1 if i is a common ancestor of j and k in the quicksort tree. That is, if both j and k appear in the subtree of $T(\pi)$ rooted at i.

Exercise 6. What is $\mathbb{E}[B_{i,j,k}]$? Give a succinct formula for this.

Exercise 7. Let $C(\pi, k)$ be the number of comparisons made by QuickSelect when given π as input. Design a formula for $C(\pi, k)$ with the help of the indicator variables $A_{i,j}$ and $B_{i,j,k}$ (analogous to the formula $\sum_{i \neq j} A_{i,j}$ for the number of comparisons made by quicksort).

Exercise 8. Suppose we use QuickSelect to find the minimum of the array. On expectation, how many comparisons will it make? Give an answer that is exact up to additive terms of order $o(n)$. You can use the fact that $H_k := 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \ln(n) + o(1)$.

Exercise 9. Derive a formula for $\mathbb{E}_\pi[C(\pi, k)]$, up to additive terms of order $o(n)$. You might want to introduce $\kappa := k/n$.