Exercise Sheet 6: Answers
COMS10018 Algorithms 2024/2025

Reminder: $\log n$ denotes the binary logarithm, i.e., $\log n = \log_2 n$.

1 Big-O Notation

Rank the following functions by order of growth: (no proof needed)

$$(\sqrt{2})^{\log n}, n^2, n!, (\log n)!, (\frac{3}{2})^n, n^3, \log^2 n, \log(n!), 2^{2^n}, n \log n$$

Hint: Stirling's approximation for the factorial function can be helpful:

$$e(\frac{n}{e})^n \leq n! \leq en(\frac{n}{e})^n$$

Solution.

$$\begin{aligned} O(\log^2 n) &\subseteq O((\sqrt{2})^{\log n}) \subseteq O(\log(n!)) \subseteq O(n \log n) \subseteq O(n^2) \\ &\subseteq O(n^3) \subseteq O((\log(n))!) \subseteq O((\frac{3}{2})^n) \subseteq O(n!) \subseteq O(2^{2^n}) \end{aligned}$$

✓

2 kth Largest Element

Give an algorithm that runs in time $O(n + k \log n)$ that computes the kth largest number in an array of n distinct integers.

Hint: Think about Heapsort!

Solution. In Heapsort, we can construct the tree in time $O(n)$. Then, we can run the first k steps of the Heapsort algorithm, which places the k largest elements at the end of the array. Each step of the sorting takes time $O(\log n)$ (which comes from the Heapify() operation). The total runtime therefore is $O(n + k \log n)$. ✓

3 Sorting

We are given an array A with $n + m$ elements so that the first n elements are sorted and the last m elements are unsorted.

1. What is the runtime of Insertionsort on array A?

Solution. $O(m(n+m))$. ✓

2. Suppose that $m = O(1)$. How can we sort A as efficiently as possible and what is the resulting runtime?

   Solution. We can run Insertionsort on the unsorted elements. This would then take time $O(n)$. ✓

3. Suppose that $m = O(\sqrt{n})$. How can we sort A as efficiently as possible and what is the resulting runtime?

   Solution. We can run any $O(m \log m)$ sorting algorithm in order to sort the unsorted elements first. Then, we merge the two sorted parts in time $O(n+m)$, resulting in a sorting algorithm that runs in time $O(m\log(m)+n+m) = O(n+m\log m)$. If $m = O(\sqrt{n})$, then the final runtime is $O(n)$. ✓

4. What is the largest value of m so that we can obtain a runtime of $O(n)$? (difficult!)

   Solution. According to the previous exercise, the runtime is $O(m\log(m)+n)$. We need to identify the largest value for m such that $O(m\log(m)+n) = O(n)$. This is equivalent to choosing the largest m such that $O(m \log m) = O(n)$.

   First, suppose that $m = \Theta(n/\log(n))$. Then:

$$\begin{aligned} m \log m &= O(n/\log(n) \cdot \log(n/\log(n))) \\ &= O(n/\log(n) \cdot (\log(n) - \log\log(n))) \\ &= O(n + n\log\log(n)/\log(n)) = O(n) \ , \end{aligned}$$

   since both n and $n\log\log(n)/\log(n)$ are in $O(n)$.

   Next, suppose that $m \in O(n)$ if $m = \Theta(f(n)n/\log(n))$, for some growing (superconstant) function f. Then:

$$\begin{aligned} m \log m &= O(f(n)n/\log(n) \cdot \log(f(n)n/\log(n))) \\ &= O(f(n)n/\log(n) \cdot (\log(f(n)) + \log(n) - \log\log(n))) \\ &= O(f(n)\log(f(n))n/\log(n) + f(n)n - f(n)n\log\log(n)/\log(n)) \notin O(n) \ , \end{aligned}$$

   since $f(n)n \notin O(n)$ (since $f(n)$ is increasing with n and hence superconstant). This implies that the largest m is in $\Theta(n/\log n)$. ✓

5. Suppose that $m = \Theta(n)$. How can we sort A as efficiently as possible and what is the resulting runtime?

   Solution. We can use any $O(n \log n)$ time sorting algorithm to obtain a total runtime of $O(n \log n)$. ✓

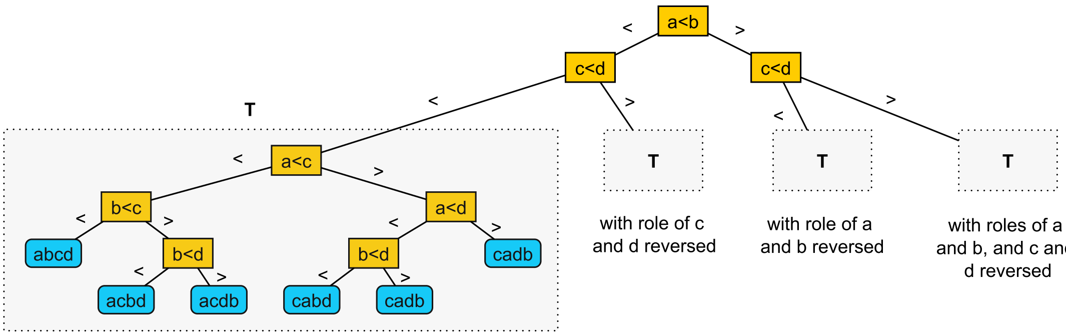
4 Decision Trees

1. Give a lower bound on the number of queries needed for sorting 4 elements.

Solution. At least 5 queries are needed. There are $4! = 24$ possible permutations, which correspond to the leaves in a decision tree. Any binary tree with 24 leaves has a height of at least 6. A root-to-leaf path of length 6 visits at least 5 internal nodes, which correspond to the number of queries. ✓

2. Give an optimal decision tree/guessing strategy for sorting 4 elements a, b, c, d (draw the decision tree).

Solution.



✓

3. How many comparisons does the Insertionsort algorithm make in the worst case when sorting an array of length 4?

Solution. In the worst case it makes 6 comparisons: In the worst case i comparisons are needed for inserting the element $A[i]$ into the already sorted prefix. Hence, we need $1 + 2 + 3 = 6$ comparisons. ✓

5 Optional and Difficult Questions

Exercises in this section are intentionally more difficult and are there to challenge yourself.

5.1 A Different Type of Sorting Algorithm

Consider the following algorithm for sorting an array A of size n:

1. Sort recursively the first 2/3 of A, i.e., $A[0, \ldots, 2/3n - 1]$
2. Sort recursively the last 2/3 of A, i.e., $A[n/3 - 1, n - 1]$
3. Sort recursively the first 2/3 of A, i.e., $A[0, \ldots, 2/3n - 1]$

Answer the following questions:

1. Argue/prove that the algorithm really sorts A.
2. What is the runtime of A?