University of Houston

Midterm 1 Review

COSC 3320
Algorithms and Data Structures

Note

Read the Academic Honesty policy.

1 Exercises

Exercise 1: Big-$\mathcal{O}$

Consider the following functions:

$$n^{1.5}, n \log n, 2^n, n!, n^n, 1/n$$

Rank the listed functions by order of growth., i.e., give an ordering f_1, f_2, …, f_5 such that $f_1 = \mathcal{O}(f_2)$, $f_2 = \mathcal{O}(f_3)$, and so on. Justify your ordering.

Exercise 2: Permutations

A permutation of a sequence $(s_0, s_1, \ldots, s_{n-1})$ is a sequence with the same terms, but in a different order. For example, the sequence (3, 4, 6) admits 6 permutations:

$$\begin{matrix} (3,\ 4,\ 6) & (3,\ 6,\ 4) \\ (4,\ 3,\ 6) & (4,\ 6,\ 3) \\ (6,\ 3,\ 4) & (6,\ 4,\ 3) \end{matrix}$$

1. Give a decrease and conquer algorithm to output all permutations of a sequence of n distinct elements.
2. Prove that this algorithm is correct.
3. Give a recurrence for the runtime of this algorithm.
4. Solve this recurrence.

Exercise 3: Hamming Weight

The *Hamming Weight* of a binary number n is the number of bits set to 1 in the binary representation of n. For example, the Hamming Weight of 14 is 3, since

$$14 = 1110_2$$

Give a divide and conquer algorithm to compute the Hamming Weight of a non-negative integer n.

Exercise 4: Recurrences

Consider the following recurrence:

$$T(n) = 3T\left(\frac{n}{2}\right) + n^2$$

1. Show that $T(n) = \mathcal{O}(n^3)$ by induction
2. Solve the recurrence using the DC Recurrence Theorem.

Exercise 5: Sorting

Consider the following sorting algorithm:

```
func SELECTION-SORT(arr):
    n = LEN(arr)
    if n ≤ 1:
        return arr
    else:
        result = SELECTION-SORT(arr[0 : n − 1])    ▷ Sort all but the last element
        last-element = arr[n − 1]
```

8: let i be the first index such that `arr`$[i-1] \leq$ `last-element` $\leq arr[i]$

9: `result`.INSERT(`last-element`, i) ▷ *Insert `last-element` into correct position*

10: **return** `result`

1. Prove the correctness of this algorithm by induction.

2. Write a recurrence, $T(n)$, for the runtime of this algorithm. Assume finding i in line 8 is done via linear search.

3. Solve this recurrence in terms of asymptotic complexity, i.e., find a function $f(n)$ such that $T(n) = \mathcal{O}(f(n))$.

4. How does this recurrence change if we find i via binary search? How does this affect the asymptotic complexity?