EC330 Applied Algorithms and Data Structures for Engineers
Spring 2021

Homework 3

Out: March 1, 2021
Due: March 12, 2021

This homework has a written part and a programming part. Both are due at 11:59 pm on March 12. You should submit both parts on Gradescope.

This is an individual assignment. See course syllabus for policy on collaboration.

1. Recurrence [30 pt]

a) Suppose you have to choose among the follow three algorithms:

- Algorithm A solves the problem by dividing it into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.
- Algorithm B solves the problem of size n by recursively solving two subproblems of size $n - 1$ and then combining the solutions in constant time.
- Algorithm C solves problems of size n by dividing them into seven problems of size $n/3$, recursively solving each subproblem, and then combining the solutions in $\boldsymbol{O}(n^2)$ time.

What are the running times of each of these algorithms (in big-$\boldsymbol{O}$ notation). Which algorithm would you choose? **[10 pt]**

b) Give the *tightest asymptotic upper bound* (in $\boldsymbol{O}(\cdot)$) you can obtain for the following recurrences. Justify your answer. **[5 pt each]**

- $T(n) = 2T(n/4) + 330\sqrt{n}$
- $T(n) = 20T(n/15) + n\log n$
- $T(n) = T(n-1) + 3\log n,\ T(1) = 1$
- $T(n) = (\log n)T(n/2) + 1$

2. Sort [20 pt]

a) Write down the recurrence relation of the following sorting algorithm. What is the time complexity of this method (in $\boldsymbol{O}(\cdot)$ notation)? Justify your answer. **[10 pt]**

```
void StrangeSort(int a[], int min, int max) {
      if (min >= max)
            return;
      if (a[min] > a[max])
            swap(a[min], a[max]);  // constant-time operation
      int one_third = (max - min + 1) / 3;
      if (one_third >= 1) {
            StrangeSort(a[], min, max - one_third);
            StrangeSort(a[], min + one_third, max);
```

```
                StrangeSort(a[], min, max - one_third);
        }
}
```

b) Write down the recurrence relation of the following sorting algorithm. What would be the best-case input for this algorithm? For that case, is it faster or slower than Bubble Sort? **[10 pt]**

```
void MysterySort(int a[], int min, int max) {
        if (min >= max)
                return;
        int mid = floor((min + max)/2);
        MysterySort(a[], min, mid);
        MysterySort(a[], mid+1, max);
        if (a[max] < a[mid])
                swap(a[max], a[mid])  // constant-time operation
        MysterySort(a[], min, max-1);
}
```

3. Programming [50 pt]

a) Implement the function *zigzagSort* in *Problem3a.cpp*. The function sorts a *singly-linked list* of integers *nums in-place* such that $nums[0] \le nums[1] \ge nums[2] \le nums[3]$ ... where $nums[i]$ is the i^{th} integer in the linked list. Below are some example inputs and the expected outputs. Note that the integers are not necessarily all distinct. Your implementation should run in $\boldsymbol{O}(n)$ time where n is the length of the linked list. **[20 pt]**

Example:
Input: 3 → 1 → 4 → 5 → 3 → 7 → 2
A possible output: 1 → 4 → 3 → 5 → 3 → 7 → 2 because $1 \le 4 \ge 3 \le 5 \ge 3 \le 7 \ge 2$
Another possible output: 1 → 5 → 3 → 7 → 3 → 4 → 2 because $1 \le 5 \ge 3 \le 7 \ge 3 \le 4 \ge 2$

b) Implement the following variant of bubble sort *twowayBubble* in *Problem3b.cpp* where we operate in both forward and backward directions, given a *doubly-linked list* of integers as input. In the first forward (rightward) pass, we will move the largest element to the end of the list. In the subsequent backward (leftward) pass, we will move the smallest element to the beginning of the list. After *i* complete passes (each complete pass consists of a forward pass and a backward pass), the invariant that we have is that the first *i* elements and the last *i* elements will be sorted (i.e. in their correct positions). **[30 pt]**

Example:
Input: 3 ↔ 5 ↔ 2 ↔ 1
After the 1st forward pass: 3 ↔ 2 ↔ 1 ↔ 5
After the 1st backward pass: 1 ↔ 3 ↔ 2 ↔ 5
We can see that 1 and 5 are now in their correct positions.