College of Computer & Information Science
Northeastern University
Fall 2018

CS5800
MS Algorithms
22 September 2018

Problem Set 2 (due Wednesday, October 3, 11:59 PM)

Instructions:

- The assignment is due at the time and date specified. Late assignments will be accepted, up until Thursday, October 4, 11:59 PM. *Note, however, that you can use at most 3 late days for your problem set and programming assignment submissions throughout the course of the term.*
- We encourage you to attempt and work out all of the problems on your own. You are permitted to study with friends and discuss the problems; however, *you must write up you own solutions, in your own words.*
- Please refrain from searching online or asking your peers or other students for solutions. The best way to learn the material is to attempt the problem yourself, and if you are stuck, identify where and why you are stuck and seek help to overcome the associated hurdles.
- If you do collaborate with any of the other students on any problem, please *list all your collaborators in your submission for each problem.*
- We require that all homework submissions be neat, organized, and *typeset*. You may use plain text or a word processor like Microsoft Word or LaTeX for your submissions. If you need to draw any diagrams, however, you may draw them with your hand.

Problem 1. (2 points) Binary Search Implementation that Passes Arrays

Consider the following implementation of the recursive binary search algorithm that passes arrays in its recursive calls.

```
BinarySearch(A, n, x):
if n = 1
  if x = A[1] return Yes
  else return No
else
  m = ⌊n/2⌋
  if x = A[m] return Yes
  if x < A[m]
    copy A[1...m] into a new array B[1...m]
    BinarySearch(B, m, x)
  else
    copy A[m+1...n] into a new array B[1...n-m]
    BinarySearch(B, n-m, x)
```

Give a recurrence for the worst-case running time of the above binary search implementation. Solve the recurrence to derive a tight bound on the running time of the above algorithm.

Problem 2. (2 points) A variant of Mergesort

Consider the following variant of Mergesort. If the array has more than two elements, it recursively sorts the first two-third and the last one-third. Then, it merges the first two-third with the last one-third. If the array has at most two elements, then it trivially sorts the array. For completeness, here is the pseudocode for sorting the array $A[\ell \ldots r]$.

```
NEWSORT(A, ℓ, r):
if r − ℓ + 1 = 1:
  exit
if r − ℓ + 1 = 2 and A[ℓ] > A[r]:
  swap A[ℓ] ↔ A[r]
  exit
if r − ℓ + 1 > 2:
  m ← ⌊(r − ℓ + 1)/3⌋
  NEWSORT(A, ℓ, ℓ + 2m − 1)        [Sort first two-third]
  NEWSORT(A, ℓ + 2m, r)            [Sort last one-third]
  MERGE(A, ℓ, ℓ + 2m − 1, r)       [Merge first two-third with last one-third]
```

Write a recurrence relation for the worst-case running time of NEWSORT. Solve the recurrence relation to obtain a Θ-bound on the running time of the algorithm, in terms of the length n of the array. You may use any of the methods to solve the recurrence. You may also ignore floors and ceilings in your calculations.

Problem 3. (4 points) Recurrences

Solve the following recurrences. You may assume for the base case that $T(1) = 1$.

(a) $T(n) = 8T(n/3) + n^2$. For convenience, you may assume that n is a power of 3.

(b) $T(n) = T(\lfloor n/3 \rfloor) + T(\lfloor n/2 \rfloor) + n^3$. For convenience, you may ignore floors and ceilings in your calculation.

Problem 4. (2 + 3 + 2 = 7 points) Finding good widgets using pairwise testing

You have n supposedly identical widgets that in principle can test each other. For instance, you can connect two widgets A and B, each of which then reports whether the other widget is good or bad, but the answer of a bad widget cannot be trusted. Thus, there are four possible outcomes of such a pairwise test.

Widget A says	Widget B says	Conclusion
B is good	A is good	both are good, or both are bad
B is good	A is bad	at least one is bad
B is bad	A is good	at least one is bad
B is bad	A is bad	at least one is bad

You are told that more than $n/2$ of the widgets are good, and you are asked to determine all of the good widgets.

(a) Consider the problem of finding a single good widget. Show that $\lfloor n/2 \rfloor$ pairwise tests are sufficient to reduce the problem to one of nearly half the size.

(b) Using part (a), give a divide-and-conquer algorithm to find a single good widget using $\Theta(n)$ pairwise tests. Give and solve the recurrence that describes the number of tests.

(c) Using part (c), show that all the good widgets can be identified using $\Theta(n)$ pairwise tests.

2. (3 + 2 = 5 points) Optimal location of store

You are a new player in the mobile phone business and are planning to open a physical store in Manhattan, to rival Apple's well-established store there. You want to determine an optimal location to place the store. Manhattan is organized as a grid, and a simple model is to view it as an $m \times n$ grid consisting of points (i, j), $1 \leq i \leq m$ and $1 \leq j \leq n$. Each of the mn points constitutes both a potential location for the store as well as a neighborhood of homes. The distance between two points (a, b) and (c, d) in Manhattan – $D((a, b), (c, d))$ – is simply $|a - c| + |b - d|$. For example, the distance between $(5, 3)$ and $(2, 9) = |5 - 2| + |3 - 9| = 9$.

After a detailed survey, you have found out which of the mn neighborhoods are really potential customers – so you have a set C of potential customers:

$$C = \{(i, j) : 1 \leq i \leq m, 1 \leq j \leq n, \text{ neighborhood at } (i, j) \text{ is a potential customer}\}.$$

For example, the set C could be $\{(1, 4), (1, 5), (1, 8), (2, 4), (3, 5), (6, 7), (2, 9)\}$. Since you have only one store to build, you would like to locate it at a point that minimizes the total distance from all the potential customers. That is, you want to find (x, y) in the grid that minimizes

$$\sum_{(i,j) \in C} D((i, j), (x, y)).$$

(a) Prove that an optimal location for the store is (x^*, y^*), where x^* is a median of $x_1, \ldots, x_n$ and y^* is a median of $y_1, \ldots, y_n$. That is, prove that for any (x, y):

$$\sum_{(i,j) \in C} D((i, j), (x^*, y^*)) \leq \sum_{(i,j) \in C} D((i, j), (x, y)).$$

(b) Using part (a), give an efficient algorithm that takes as input the $m \times n$ grid and the set C, and returns the optimal store location (x, y).