College of Computer & Information Science
Northeastern University
Fall 2018

CS5800
MS Algorithms
5 October 2018

Sample Solution to Problem Set 2

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.

Answer: The algorithm takes time at most $n/2$ for copying half of the array to B, and constant time for all of the other non-recursive steps. The recursion is on an array of half the size. So we obtain a recurrence of the following form.

$$T(n) \leq T(\lceil n/2 \rceil) + n,$$

with $T(1) = 1$. This leads to the solution $T(n) = O(n)$. The copying step itself takes linear time, so we also have $T(n) = \Omega(n)$ leading to the bound $T(n) = \Theta(n)$.

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.

Answer: We get the following recurrence.

$$T(n) = T(\lfloor 2n/3 \rfloor) + T(\lceil n/3 \rceil) + \Theta(n).$$

If we ignore floors and ceiling and use the recursion tree approach, each level contributes $\Theta(n)$ until level $\log_3 n$. There are at most $\log_{3/2} n$ levels. So we obtain $T(n) \leq n \log_{3/2} n$ and at least $n \log_3 n$. So $T(n) = O(n \log n)$.

We can also prove using substitution method and induction.

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.

Answer: We can use the recursion tree approach.

First level is n^2. The second level is $8n^2/9$. The third is $8^2n^2/9^2$. In general, the ith level is $8^i n^2/9^i$. This is a geometrically decreasing sequence. The number of levels is $\log_3 n$. But even if we take the geometric series to an infinite levels, we have:

$$T(n) \leq n^2(1 + 8/9 + 8^2/9^2 + \ldots) = 9n^2.$$

We thus have $T(n) = O(n^2)$. Clearly, $T(n) \geq n^2$. So $T(n) = \Theta(n^2)$.

We can also use the Master Theorem.

$$n^{\log_b a} = n^{\log_3 8} \quad f(n) = n^2.$$

Since $\log_3 8 < 2$, for any $0 < \varepsilon \leq 2 - \log_3 8$, we have $n^{\log_b a + \varepsilon} = O(f(n))$. Therefore, $T(n) = \Theta(f(n))$.

(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.

Answer: We solve this recurrence using the recursion tree approach.

First level is n^2. The second level is $(n/3)^3 + (n/2)^3 = 35n^3/216$. The third level is $(35/216)^2 n^3$. In general, the ith level is $(35/216)^{i-1} n^3$. This is a geometrically decreasing sequence.

Even if we take the geometric series to an infinite levels, we have:

$$T(n) \leq n^3(1 + 35/216 + 35^2/216^2 + \ldots) = 216n^3/181.$$

Therefore, $T(n)$ is $O(n^3)$. Clearly, $T(n) \geq n^3$. So $T(n)$ is also $\Omega(n^3)$. Hence, $T(n)$ is $\Theta(n^3)$.

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.

(**Note:** This problem was taken from Cormen-Leiserson-Rivest-Stein.)

Answer of part (a): We will prove the desired claim under the assumption that the number of good and bad widgets are the same. If the number of bad widgets is greater than the number of good widgets, then this makes the task of finding a good widget only harder.

Now let G be the set of good widgets and B be the set of bad widgets. Suppose each bad widget reports a good widget as bad and a bad widget as good. Then we have the following property: (i) all the pairwise tests within G report good-good, (ii) all pairwise tests within B report good-good, and (iii) all pairwise tests involving one widget in G and the other in B report bad-bad. This implies that after doing all the pairwise tests, the professor will not be able to distinguish the set G from B since their behavior is exactly the same, and so is their number. Thus, even though the professor may be able to identify that all of the widgets in G are either all good or all bad and the widgets in B are all good or all bad, he or she will not be able to identify a single good widget.

Answer of part (b): Let S be the given set of n widgets. Let g be the number of good widgets and b be the number of bad widgets. We group the given set S widgets into pairs. If n is odd, then

there may be an unpaired widget. Thus, we have $\lfloor n/2 \rfloor$ pairs of two widgets each. We load each pair into the jig and perform the pairwise test.

We will construct a set S' of at most $\lceil n/2 \rceil$ widgets such that at least half of them are good. We go through the results of the pairwise tests. If the test result is good-bad, then we throw out the widget that tested bad and add to S' the widget that tested good. If the test result is good-good, then we add to S' any one of the widgets and throw the other. If the test result is bad-bad, then we throw out both the widgets. Since, at most widget from each pair is included in S', it is clear that the number of widgets in S' is at most $\lfloor n/2 \rfloor$. Finally, if the number of widgets thus included is an even number, then we add the unpaired widget to S'. We thus have $|S'| \leq \lceil n/2 \rceil$.

Let gg, bb, and gb be the number of pairwise tests involving two good widgets, two bad widgets, and one good widget and a bad widget, respectively. (Note that the terms "good" and "bad" in the preceding sentence refer to the actual quality of the widget, not the outcome of the test.) Let x and y denote the number of good widgets and bad widgets, respectively, that were picked during the pairwise tests. When two good widgets pair together, the output of the test is GG; therefore at least one of the widgets is included in S'. If a good widget and a bad widget pair together, then the result is either a GB or a BB. In the former case, only the good widget is picked. In the latter case, both widgets are discarded. Finally, when two bad widgets are paired together, at most one of the widgets is selected. Thus, we have $x \geq gg$ and $y \leq bb$.

Since the number of good widgets is at least $n/2$, we have $gg > bb$ if n is even or if n is odd and the unpaired widget is bad. If n is odd and the unpaired widget is good, then $gg \geq bb$. We thus have $x \geq y$. We still have to account for the unpaired widget, which arises only if n is odd. Recall that we pick the unpaired widget only if the number of widgets selected after the pairwise tests is even. If $x + y$ is odd, then since $x \geq y$, it follows that $x > y$ and we are done in this case. If $x + y$ is even, then we have two cases: either $x = y$ or $x \geq y + 2$. The former case can only arise if $x = gg = bb = y$. And $gg = bb$ can only happen if the unpaired widget is good, which makes the number of good widgets in S' one more than the number of bad widgets. Finally, if $x \geq y + 2$, then adding another widget still leaves the good widgets with a clear majority. This completes the proof.

Answer of part (c): We first identify a good widget by using the procedure of part (b) recursively. The recursion for this procedure is given by

$$T(n) = T(\lceil n/2 \rceil) + \lfloor n/2 \rfloor.$$

Ignoring the ceiling, we obtain the recurrence $T(n) \leq T(n/2) + n/2$, which can be shown to be $O(n)$ using the recursion tree method, as we have done several times in class.

Once we have identified a good widget, we test the widget with each of the other $n - 1$ widgets and identify their quality. This procedure takes $n - 1 = \Theta(n)$ tests. Thus the total number of pairwise tests is $\Theta(n)$.

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):

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

Answer: We show 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 $y_1, \ldots, y_n$. We renumber the x_i's and y_i's in sorted order. Let $x_m = x^*$ be the median of the x_i's and $y_p = y^*$ be the median of the y_i's. We now show that

$$\sum_{i=1}^{n} |x_i - x_m| \leq \sum_{i=1}^{n} |x_i - x|,$$

for any x.

We first consider the case when $x < x_m$. In this case, we have

$$\begin{aligned}
\sum_{i=1}^{n}(|x_i - x_m| - |x_i - x|) &\leq \sum_{x_i<x_m} |x_m - x| - \sum_{x_i\geq x_m} |x_m - x| \\
&= |x_m - x| \left(\sum_{x_i<x_m} - \sum_{x_i\geq x_m} \right) \\
&= |x_m - x| \left((2 \sum_{x_i<x_m} 1) - 1 \right) \\
&\leq 0.
\end{aligned}$$

Similarly, if $x > x_m$, we have

$$\begin{aligned}
\sum_{i=1}^{n}(|x_i - x_m| - |x_i - x|) &\leq - \sum_{x_i\leq x_m} |x_m - x| + \sum_{x_i>x_m} |x_m - x| \\
&= |x_m - x| \left((2 \sum_{x_i>x_m} 1) - 1 \right) \\
&\leq 0.
\end{aligned}$$

By the same argument, we also have

$$\sum_{i=1}^{n} |y_i - y_p| \leq \sum_{i=1}^{n} |y_i - y|,$$

for any y. This shows that (x_m, y_p) is an optimal location for the store.

(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).

Answer: Both the coordinates x_m and y_p can be found using the linear time median algorithm we studied in class. Therefore, the optimal store location can be found in linear time.