Homework #2
Introduction to Algorithms
601.433/633
Spring 2020

Due on: Tuesday, February 13th, 12pm
Format: Please start each problem on a new page.
Where to submit: On Gradescope, please mark the pages for each question

February 23, 2020

1 Problem 1 (12 points)

Given a list of n integers $x_1, \ldots, x_n$ (possibly negative), find the indices $i, j \in [n]$ such that $x_i \cdot x_j$ is maximized. Your algorithm must run in $O(n)$ time.

Proof. The algorithm can be described as follows

1. If $n \leq 2$, output the product of the numbers.
2. Else, let $m_p, m_n := 0$. Find the two largest positive numbers, if they exist, and set m_p to be their product. Similarly, find the two smallest negative numbers, if they exist, and set m_n to be their product.
3. Output $\max(m_p, m_n)$.

Proof of Correctness: If $n \leq 2$, we simply multiply the two numbers. Otherwise there must exist two numbers with the same sign so the answer is non-negative. Since positive integers are larger than negative ones, we only need to choose two numbers with the same sign. Since $f(x, y) = xy$ is monotonic in x and y, the largest two positive numbers and the smallest two negative numbers will have the largest product among positive and negative integers respectively.

Proof of Runtime: We need to loop through all n integers once to find the largest and smallest integers. Since we do a constant amount of work in each iteration of the loop, the runtime is $O(n)$. $\square$

2 Problem 2 (12 points)

Let S be an array of integers $\{S[1], S[2], \ldots, S[n]\}$ such that $S[1] < S[2] < \cdots < S[n]$. Design an algorithm to determine whether there exists an index i such at $S[i] = i$. For example, in $\{-1, 2\}$, $S[2] = 2$.

Your algorithm should work in $O(\log n)$ time. Prove the correctness of your algorithm.

Proof. We output SpecialBinSearch(S, 1, $n + 1$).

Algorithm 1 SpecialBinSearch(X, i_s, i_l)

```
if |i_l − i_s| == 1 then
    return X[i_s] == i_s
else if X[⌊(i_l+i_s)/2⌋] ≤ ⌊(i_l+i_s)/2⌋ then
    return SpecialBinSearch(X, ⌊(i_l+i_s)/2⌋, i_l)
else
    return SpecialBinSearch(X, i_s, ⌊(i_l+i_s)/2⌋)
end if
```

Proof of Correctness:

Claim: For any array $X = X[1] < \cdots < X[n]$ of integers, the array $\{X[i] - i : i \in [n]\}$ is non-decreasing. *Proof*: Since $X[i] < X[i + 1]$ for all $i \in [n - 1]$, $X[i] - i < X[i + 1] - i \implies X[i] - i \leq X[i + 1] - (i + 1)$ for all $i \in [n - 1]$ because $\{X[i] : i \in [n]\}$ are integers.

To prove correctness of our algorithm [1], we induct on the quantity $i_l - i_s$. We're going to prove that SpecialBinSearch(X, i_s, i_l) correctly outputs if there's any index $j \in [i_s, i_l]$ such that $X[j] = j$ when $i_l - i_s \leq n$ and X is sorted in increasing order. If we prove this then we have proved that SpecialBinSearch(S, 1, $n + 1$) is correct.

[1] There are many ways to prove correctness. You could induct on many things, including the sequence of arrays that the algorithm recurses on etc.

- Base Case, $i_l - i_s = 1$: Since there's only element, the algorithm correctly outputs $X[i_s] == i_s$.
- Inductive Hypothesis (IH): We assume that SpecialBinSearch(X, i_s, i_l) correctly outputs if there's any index $j \in [i_s, i_l]$ such that $X[j] = j$ when $i_l - i_s < n$.
- Inductive Step, $i_l - i_s = n$: If there exists $i \in [i_s, i_l]$ such that $X[i] = i$, then the value of the i^{th} index in $\{X[i] - i : i \in [n]\}$ is 0.

  Case $i \geq \lfloor \frac{i_l + i_s}{2} \rfloor$: This means that $X[\lfloor \frac{i_l+i_s}{2} \rfloor] - \lfloor \frac{i_l+i_s}{2} \rfloor \leq 0$ because the array $\{X[i] - i : i \in [n]\}$ is sorted (by the proved claim). We can then invoke the IH on SpecialBinSearch(X, $\lfloor \frac{i_l+i_s}{2} \rfloor$, i_l) since $i_l - \lfloor \frac{i_l+i_s}{2} \rfloor < n$.

  Case $i < \lfloor \frac{i_l + i_s}{2} \rfloor$: This means that $X[\lfloor \frac{i_l+i_s}{2} \rfloor] - \lfloor \frac{i_l+i_s}{2} \rfloor > 0$ because the array $\{X[i] - i : i \in [n]\}$ is sorted (by the proved claim). We can then invoke the IH on SpecialBinSearch(X, i_s, $\lfloor \frac{i_l+i_s}{2} \rfloor$) since it has length less than $|S|$.

Proof of Runtime: In an iteration with input X, the algorithm recurses on array of size at most $|X|/2$ and does a constant amount of work in that iteration. Hence the runtime $T(n)$ of an input S of size n can be written as the recurrence $T(n) \leq T(n/2) + c$. We can solve this as

$$
\begin{aligned}
T(n) &\leq T(n/2) + c \\
&= \sum_{i=1}^{\log_2(n)} c = c\log(n) = O(\log(n))
\end{aligned}
$$

□

3 Problem 3 (13 points)

We say a 3-tuple of positive real numbers (x_1, x_2, x_3) is *legal* if a triangle can have sides of length x_1, x_2 and x_3. Given a list of n positive real numbers $\{x_1, \ldots, x_n\}$, count the number of unordered 3-tuples (x_i, x_j, x_k) that are legal. For example, for the numbers $\{3, 5, 8, 4, 4\}$, $(3, 4, 5)$ is a legal tuple while $(4, 4, 8)$ is not.

Your algorithm should run in $O(n^2 \log(n))$ time. Prove correctness of your algorithm.

Proof. The following algorithm counts the number of legal unordered 3-tuples in a list of reals.

$A_S \leftarrow$ list of reals sorted in ascending order.
$c \leftarrow 0$
for p_S in $[n]$ **do**
  for p_L in $[p_S + 1, n]$ **do**
    $x_S \leftarrow A_S[p_S]$
    $x_L \leftarrow A_S[p_L]$
    $c \leftarrow c+$ number of values from A_S in interval $[\max(x_S, x_L - x_S), x_S + x_L]$
    $\{$Use binary search to find the indices of the interval$\}$
  end for
end for
return $c/2$

Proof of correctness [2]**:** Let (x_i, x_j, x_k) be a legal triangle. W.l.o.g we will assume that $i < j < k$ **w.r.t to** A_S.

We claim that the algorithm counts (x_i, x_j, x_k) twice. When $p_S = i$ and $p_L = j$ then since $x_i + x_j \geq x_k \geq x_i, x_j$ (by the triangle inequality and the fact that A_S is sorted), x_k must lie in the interval $[\max(x_i, x_j - x_i), x_i + x_j]$. When $p_S = i$ and $p_L = k$ then $x_k - x_i \leq x_j$ by the triangle inequality and $x_i \leq x_j \leq x_k \leq x_k + x_i$, hence x_j must lie in the interval $[\max(x_i, x_k - x_i), x_i + x_k]$. Since we count every triangle by setting p_S and p_L and p_S is always the smallest edge, these are the only settings we can count (x_i, x_j, x_k).

Now, let us prove that we don't count illegal tuples. Assume for sake of contradiction that (x_i, x_j, x_k) is a tuple that we count that is not legal. W.l.o.g assume that this happens when $p_S = i$, $p_L = j$ then $x_k \in [\max(x_i, x_j - x_i), x_i + x_j]$ which means $x_k \geq x_j - x_i$ and $x_k \leq x_i + x_j$. But these satisfy all the triangle inequalities and hence is a contradiction to the fact that (x_i, x_j, x_k) is not legal.

Proof of runtime: Sorting the array takes $O(n \log n)$ time. There are n iterations of the outer loop, and n iterations of the inner loop for each iteration of the outer loop. In each iteration of the inner loop we do $O(\log n)$ work from the binary search. This gives a total of $O(n^2 \log n)$ runtime for the algorithm.

□

4 Problem 4 (13 points)

You are given one unsorted integer array A of size n. You know that A is almost sorted, that is it contains at most m inversions, where inversion is a pair of indices

[2] We can prove this using induction too but for sake of exposure let's try a more direct proof.

(i, j) such that $i < j$ and $A[i] > A[j]$.

1. To sort array A you applied algorithm Insertion Sort. Prove that it will take at most $O(n + m)$ steps.

   Solution:
   To prove this fact, consider the pseudocode for insertion sort.

Algorithm 2 Insertion-Sort

```
for j := 2 to length[A] do
    key := A[j]
    i := j − 1
    while i > 0 and A[i] > A[i + 1] do
        A[i + 1] := A[i]
        A[i + 1] := key
    end while
end for
```

   Everything except the while loop requires $\Theta(n)$ time. We now observe that every iteration of the while loop can be thought of as swapping an adjacent pair of outof-order elements $A[i]$ and $A[i + 1]$. Such a swap decreases the number of inversions in A by exactly one since $(i, i + 1)$ will no longer be an inversion and the other inversions are not affected. Since there is no other means of increasing or decreasing the number of inversions of A, we see that the total number of iterations of the while loop over the entire course of the algorithm must be equal to m.

2. What is a maximum possible number of inversions in the integer array of size n?

   Solution:
   Maximum number of inversion is equal to $\frac{n(n-1)}{2}$. It happens if every pair of items is an inversion, in total we have $\frac{n(n-1)}{2}$ pairs. It happens when we array is inverse sorted.