CSE 6140 / CX 4140 Assignment 1
due Sep 4, 2020 at 11:59pm on Canvas

Please type your answers (LaTeX highly recommended) and upload a single PDF including all your answers. If you want to draw a graph by hand you can take a picture of your drawing and insert it to your PDF file. Please make sure that your inserted picture can be clearly read. Do not forget to acknowledge your collaborators.

1 Simple Complexity (11 pts)

1. (6 pts) For each pair of functions f and g, write whether f is in $\mathbb{O}(g)$, $\Omega(g)$, or $\Theta(g)$.
   (a) $f = (n + 1000)^4$, $g = n^4 - 3n^3$
   (b) $f = \log_{1000} n$, $g = \log_2 n$
   (c) $f = n^{1000}$, $g = n^2$
   (d) $f = 2^n$, $g = n!$
   (e) $f = n^n$, $g = n!$
   (f) $f = \log n!$, $g = n \log n$ (hint: Stirling's approximation)

   solution:
   (a) $f \in \Theta(g)$
   (b) $f \in \Theta(g)$
   (c) $f \in \Omega(g)$
   (d) $f \in \mathbb{O}(g)$
   (e) $f \in \Omega(g)$
   (f) $f \in \Theta(g)$ since by Stirling's approximation formula, $\log n! = n \log n - n + \mathbb{O}(\log n)$.

2. (5pts) Determine the Big-O time complexity for the algorithm below (**show your analysis**). Also, very briefly explain (in one or two sentences) what the algorithm outputs (note: the % symbol is the modulo operator):

```
   Data: n
1  i = 1;
2  while i ≤ n do
3  |  j = 0;
4  |  k = i;
5  |  while k % 3 == 0 do
6  |  |  k = k/3 ;
7  |  |  j++ ;
8  |  end
9  |  i++ ;
10 |  print i,j;
11 end
```

Solution: The Big- O time complexity for this algorithm is $\mathbb{O}(n \log n)$ since the outer while lop takes n steps and the inner while loop takes at most $\log i$ steps. Each output pair i, j tells us that in the prime factorization of $(i-1)!$, 3 is raised to power j.

2 Algorithm design and complexity (15 pt)

The problem consists of finding the lowest floor of a building from which a box would break when dropping it. The building has n floors, numbered from 1 to n, and we have k boxes. There is only one way to know whether dropping a box from a given floor will break it or not. Go to that floor and throw a box from the window of the building. If the box does not break, it can be collected at the bottom of the building and reused.

The goal is to design an algorithm that returns the index of the lowest floor from which dropping a box will break it. The algorithm returns $n+1$ if a box does not break when thrown from the n-th floor. The cost of the algorithm, to be kept minimal, is expressed as the number of boxes that are thrown (note that re-use is allowed).

1. For $k \geq \lceil \log(n) \rceil$, design an algorithm with $O(\log(n))$ boxes thrown.

   Solution: The idea is to use binary search algorithm. Let m be the index of the lowest floor from which dropping a box will break it. Our goal is to find m.

   - Initially, we know that $low \leq m \leq high$, where $low = 0$ and $high = n$.
   - Then, we calculate $mid = \lfloor \frac{low+high}{2} \rfloor$ and go to the midth floor and throw a box there. If the box breaks, we update $high = mid$. Otherwise, we update $low = mid + 1$. We repeat this step as long as $low < high$.
   - Finally, we will reach to a point when $low = high$. This is when the algorithm stops and we will find $m = low (= high)$.

   This algorithm has $O(\log(n))$ boxes thrown because each time after we throw a box we half the length of interval we are searching m from.

2. For $k < \lceil \log(n) \rceil$, design an algorithm with $O\left(k + \frac{n}{2^{k-1}}\right)$ boxes thrown.

Solution: Here we first use binary search algorithm and then use linear search from bottom to top. Again, let m be the index of the lowest floor from which dropping a box will break it. Our goal is to find m.

- Initially, we know that $low \leq m \leq high$, where $low = 0$ and $high = n$.
- Then, we calculate $mid = \lfloor \frac{low+high}{2} \rfloor$ and go to the midth floor and throw a box there. If the box breaks, we update $high = mid$. Otherwise, we update $low = mid + 1$. We repeat this binary search step for $k - 1$ times.
- Now we have thrown $k - 1$ boxes and are left with one box. Moreover, we know that $low \leq m \leq high$ with $high - low = O(\frac{n}{2^{k-1}})$. We will use linear search algorithm. We go to the lowth floor and throw the box. If the box dose not break, we update $low = low + 1$ and repeat this step. If the box breaks, our algorithm stops here and we have found that $m = low$.

This algorithm has $O\left(k + \frac{n}{2^{k-1}}\right)$ boxes thrown because we throw $k-1$ times during the binary search and throw at most $O(\frac{n}{2^{k-1}})$ times during the linear search.

3. For $k = 2$, design an algorithm with $O(\sqrt{n})$ boxes thrown.

Solution: In this case, we apply linear search in two different procedures. Again, let m be the index of the lowest floor from which dropping a box will break it. Our goal is to find m.

- Firstly, we let $a = \lfloor \sqrt{n} \rfloor$, go to the ath floor and throw the first box. If the box dose not break, we update $a = a + \lfloor \sqrt{n} \rfloor$ and repeat the previous step until the the first box breaks. At this point, after at most $\lceil \sqrt{n} \rceil$ box throws, we know that $a - \lfloor \sqrt{n} \rfloor + 1 \leq m \leq a$.
- Now we use linear search to find m from the interval $[low, high]$ with $low = a - \lfloor \sqrt{n} \rfloor + 1$ and $high = a$. We go to the lowth floor and throw the second box. If the box dose not break, we update $low = low + 1$ and repeat this step. If the box breaks, our algorithm stops here and we have found that $m = low$. There are at most $\lfloor \sqrt{n} \rfloor$ thrown we need to make here.

This algorithm has $O(\sqrt{n})$ boxes thrown since the maximum number of thrown we need is $2\lceil \sqrt{n} \rceil$.

Please explain your algorithms clearly.

3 Greedy - points on a 2D plane (12pt)

You are given n distinct points and one line l on the plane and some constant $r > 0$. Each of the n points is within distance at most r of line l (as measured along the perpendicular). You are to place disks of radius r centered along line l such that every one of the n points lies within at least one disk. Devise a greedy algorithm that runs in $O(n \log n)$ time and uses a minimum number of disks to cover all n points; prove its optimality.

Solution: Without loss of generality, let us assume that the line l is the x-axis. (Otherwise, we can set l as x-axis and apply a change of coordinates to update the coordinates of the n points in our new coordinate system.) Each given point $p_i = (x_i, y_i)$, $i = 1, ..., n$, corresponds to an interval $I_i = [x_i - \sqrt{r^2 - y_i^2}, x_i + \sqrt{r^2 - y_i^2}] := [a_i, b_i]$ on l. As long as we put the center of a disk of radius r inside I_i, this disk will cover p_i. So the problem can be transformed to find the minimum number of center points for disks in l such that each interval contains at least one points. We can use the following greedy algorithm to do the job for us by putting each disk to cover as many uncovered points as possible.

Data: $(a_i, b_i), i = 1, 2, ..., n$
Result: d, C
Sort intervals by the left endpoints so that $a_1 \leq a_2 \leq ... \leq a_n$;
d = 0;
$minb = b_1$;
C = Φ (empty set);
for *i = 1 to n* **do**
  if $a_i \leq minb$ **then**
    $minb = \min(b_i, minb)$
  else
    put *minb* into C ;
    $minb = b_i$;
    d = d + 1;
  end
  put *minb* in to C;
  d = d + 1
end

The outputs d gives us the minimum number of disks and C is a set of d centers of the disks we selected to cover all the given points in the plane. Now let us use greedy stays ahead argument to prove that the greedy algorithm we designed is optimal.

- Let $A : a_1, a_2, ..., a_k$ denote the set of disks selected by greedy.
- Let $O : o_1, o_2, ..., o_m$ denote the set of disks in the optimal solution.

Let's label the points in the plane from left to right and let $f(a)$ be the maximum index of the points covered by disk a. Then we have

$$f(a_1) < f(a_2) < ... < f(a_k).$$

We also order the optimal solution in that way

$$f(o_1) < f(o_2) < ... < f(o_k).$$

Claim: For all indices $r \leq k$, $f(o_r) \leq f(a_r)$.
Proof: We prove the claim by induction.
Base case: when $r = 1$, $f(o_1) \leq f(a_1)$ is true by greedy choice.
Inductive hypothesis: $f(o_{r-1}) \leq f(a_{r-1})$.
Inductive step: We have $f(o_{r-1}) \leq f(a_{r-1}) < f(a_r)$. If $f(o_r) > f(a_r)$, then disk o_r covers points with indices $f(o_{r-1}) + 1, ..., f(a_{r-1}), ..., f(a_r), ..., f(o_r)$. This is impossible because otherwise the greedy algorithm will select disk a_r to covers points with indices $f(a_{r-1}), ..., f(a_r), ..., f(o_r)$. So $f(o_{r-1}) \leq f(a_{r-1}) < f(a_r)$.

Now let us prove that the greedy algorithm is optimal ($k = m$). Suppose $k > m$, then we have

$$n = f(o_m) \leq f(a_m) < f(a_k) = n.$$

This is a contradiction! So we have $k = m$ and the greedy algorithm is optimal.

4 Greedy - Why is the pool always so busy anyway? (12 pt)

Georgia Tech is trying to raise money for the Technology Square Research Building and has decided to host the "Tech Swim Run Bike" (TSRB) Triathalon to start off the fundraising campaign!

Usually in a triathalon all athletes will perform the three events in order (swimming, running, then biking) asynchronously, but unfortunately due to some last minute planning, the race committee was only able to secure the use of one lane of the olympic pool in the campus recreation center. This means that there will be a bottleneck during the first portion (the swimming leg) of the race where only one person can swim at a time. The race committee needs to decide on an ordering of athletes where the first athlete in the order will swim first, then as soon as the first athlete completes the swimming portion the next athlete will start swimming, etc. The race committee, not wanting to wait around for an extremely long time for everyone to finish, wants to come up with a schedule that will minimize the time it takes for everyone to finish the race. Luckily, they have prior knowledge about how fast the athletes will complete each portion of the race, and they have you, an algorithm-ista, to help!

Specifically, for each athlete, i, they have an estimate of how long the athlete will take to complete the swimming portion, s_i, running portion, r_i and biking portion, b_i. A schedule of athletes can be represented as a list of athletes (e.g. $[athlete_7, athlete_4, ...]$) that indicate in which order the athletes will start the race. Using this information, they want you to find an ordering for the athletes to start the race that will minimize the time taken for everyone to finish the race, assuming all athletes perform at their estimated time. More precisely, give an efficient algorithm that produces a schedule of athletes whose completion time is as small as possible and prove that it gives the optimal solution using an exchange argument.

Keep in mind that once an athlete finishes swimming, they can proceed with the running and biking portions of the race, even if other athletes are already running and biking. Also note that all athletes have to swim first (i.e. some athletes won't start off running or biking). For example: if we have a race with 3 athletes scheduled to go in the order $[2, 1, 3]$, and the finishing time for athlete i is represented as a_i, then the finishing times would be as follows: (with a total finishing time of $\max(a_1, a_2, a_3)$)

$$a_2 = (s_2) + r_2 + b_2$$
$$a_1 = (s_2 + s_1) + r_1 + b_1$$
$$a_3 = (s_2 + s_1 + s_3) + r_3 + b_3$$

Solution: Greedy algorithm: choose the next athlete with the longest time for running and biking.

Data: $(s_i, r_i, b_i), i = 1, 2, ..., n$

Sort data by $r_i + b_i$ so that $r_1 + b_1 \geq r_2 + b_2 \geq ... \geq r_n + b_n$;

for *i = 1 to n* **do**

  let $athlete_i$ start swimming

end

The result of this greedy algorithm gives us an ordering for athletes to start the race that will minimize the time taken for all of them to finish the race. Now let us use exchange argument to prove that the greedy algorithm is optimal.

Denote greedy solution as G, and an optimal solution O.

- Compare Solutions. Greedy solution has no inversion, namely, $athlete_i$ starts before $athlete_j$, i.e. $i < j$, if and only if $r_i + b_i \geq r_j + b_j$. If $G \neq O$, then O must have at least one inversion. Next, we will prove that solution O can be gradually converted into G without hurting the quality of O.

- Exchange Pieces. First, the argument in the proof of Observation 4 in Slides-08-25 tells us that O has at least one adjacent inversion. That is, there exists $i < j$ such that $r_i + b_i \leq r_j + b_j$. Now we claim that exchanging these two adjacent, inverted jobs i and j reduces the number of inversions by 1 and dose not increase the max time taken for every athlete. Let T be the total finishing time, i.e. the maximum of the finishing times $T_k, k = 1, 2, ..., n$, for all the athletes before the swap, and let T' be it afterwards.
  Firstly we have $T_k = T'_k$ for all $k \neq i, j$.
  Secondly, $T'_j = s_1 + s_2 + ... + s_j + r_j + b_j < s_1 + s_2 + ... + s_i + s_j + r_j + b_j = T_j$.
  And thirdly we have $T'_i = s_1 + s_2 + ... + s_j + s_i + r_i + b_i \leq s_1 + s_2 + ... + s_i + s_j + r_j + b_j = T_j$.
  So we have $T' \leq T$ and thus swapping two adjacent, inverted jobs i and j reduces the number of inversions by 1 and dose not increase the total finishing time.

- Iteration. Since there are at most $n(n-1)/2$ inversions in O, we can swap the adjacent inversions finitely many times and convert O to G without hurting the quality of O.

Therefore, our greedy algorithm is optimal.