

CS375 Weekly Assignment 2 Solutions

Due Feb 24th 10PM ET

A general note for CS375: When writing up your homework, please write neatly and explain your answers clearly, giving all details needed to make your answers easy to understand. Graders may not award full credit to incomplete or illegible solutions.

You are encouraged to discuss the problems with your classmates, **but your submission should be your own**. When writing your solutions up, you should not be looking at any other student's submission.

Clear communication **is the point**, on every assignment. In general in CS375, unless explicitly specified otherwise, answers should be accompanied by explanations. Answers without explanations may not receive full credit. Please feel free to ask me any questions about explanations that might come up!

Necessary acknowledgements: many of these questions are reproductions or adaptations of questions of Eric Aaron's devising. Others are from our course texts as listed on the syllabus.

1. Suppose we know that algorithm A has worst-case runtime of $O(n^2)$ and a best-case runtime of $O(n \log n)$.
 - (a) Could A 's best-case runtime be $\Omega(n^2)$? Explain your answer.
Solution: It could not - we know that the best-case runtime is asymptotically *upper-bounded* by $n \log n$, and $n \log n \neq \Omega(n^2)$.
 - (b) Could A 's best-case runtime be $\Theta(n)$? Explain your answer.
Solution: Yes - all we know is that its best case runtime is asymptotically upper-bounded by $n \log n$, and $n = O(n \log n)$.
 - (c) Could A 's worst-case runtime be $\Theta(n^3)$? Explain your answer.
Solution: No, we know that the worst case runtime is asymptotically upper-bounded by n^2 and $n^3 \neq O(n^2)$.
 - (d) Could A 's worst-case runtime be $\Theta(1)$? Explain your answer.
Solution: Yes! Again, all we know are some upper bounds for both the best and worst case runtimes - this means both best and worst case could be as low as constant.

2. In order to get through the project, we need to talk about brute-force algorithms. In particular, you really ought to know and understand the words **(sub)set** and **permutation**.

(a) A **set** A of items is a collection of items without any repeats. A **subset** B of A is any set of items all of which belong to A . For example, if $A = \{1, 2, 3\}$ and $B = \{2, 3\}$, then B is a subset of A .

i. Write out the 8 subsets of $\{1, 2, 3\}$:

Solution: $\{\}, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}$.

ii. For a given set of n items, how many distinct subsets are there?

Solution: 2^n - a way to think about this is that any particular subset is a specific of a binary choice per item: whether it is in the subset or not. As a result, there's 2 choices per item, giving 2^n different combinations in total.

iii. If you haven't seen it before, you should also know the following: given a set A of n items, the number of distinct ways to write a subset of A of size $1 \leq k \leq n$ is

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{\prod_{j=1}^k (n-j+1)}{\prod_{j=1}^k j} \leq n^k$$

iv. The following is pseudo-code for generating a list of all the subsets of a given array of items:

```
def GenerateAllSubsets(A):
    output = [[]] # A list containing an empty list
    for x in A:
        newList = [] # Another new list
        for lst in output:
            add lst to newList
            copyList = copy(lst) # A copy of list
            add x to copyList
            add copyList to newList
        output = newList
    return output
```

Analyze the code to show that its runtime is $\Theta(n2^n)$ (both worst and best case).

Solution: So this was kinda my bad. I definitely think you can show that the upper bound is $n2^n$ by thinking about the following: the runtime of each iteration of the outer for loop is the time it takes to iterate through each subset in `output` and make a copy it (I'm assuming a reasonable data structure that will let me add to it in constant time¹). Note that each subset during the i th iteration (starting from $i = 0$, ending at $i = n - 1$) of the outer for loop is of size *at most* i ; additionally, since `output` contains at that time all subsets of the first i items in A , we get that `output`'s size is 2^i . This means that the runtime of this code is upper bounded by $O(\sum_{i=0}^{n-1} 2^i i)$. Of course, $i < n$ in that sum, so this is also upperbounded by $O(\sum_{i=0}^{n-1} 2^i n) = O(n \sum_{i=0}^{n-1} 2^i) = O(n2^n)$ (that last inequality follows from one of the induction problems on the last weekly assignment).

Getting a tight lowerbound on the runtime is a bit trickier and definitely out of the bounds of this class. To be honest, I'd be happy seeing any lowerbound that is at least $\Omega(2^n)$, which you could get by observing that there are 2^n subsets in `output` at the end of the day, which obviously we need to generate each of. Here's the 'correct'

¹or at least constant time on average

way to do it anyway:

$$\begin{aligned}
\text{the runtime of the above alg} &\geq \text{the runtime of the last iteration of the outer for loop} \\
&\geq \sum_{\substack{\text{subsets } S \\ \text{in output}}} |S| \\
&\geq \sum_{k=0}^{n-1} \sum_{\substack{\text{subsets } S \\ \text{in output} \\ \text{of size } k}} |S| && \text{this just breaks the sum into smaller parts} \\
&= \sum_{k=0}^{n-1} k \sum_{\substack{\text{subsets } S \\ \text{in output} \\ \text{of size } k}} 1 \\
&= \sum_{k=0}^{n-1} k \left(\begin{array}{l} \text{the number of ways} \\ \text{to pick a set of size } k \\ \text{from } n \text{ items} \end{array} \right) \\
&= \sum_{k=0}^{n-1} k \binom{n}{k} && \text{from the last assignment} \\
&= \sum_{k=0}^{n-1} k \frac{n!}{k!(n-k)!} \\
&= \sum_{k=1}^{n-1} n \frac{(n-1)!}{(k-1)!(n-k)!} \\
&= n \sum_{k=1}^{n-1} \binom{n-1}{k-1} \\
&= n \sum_{k=0}^{n-2} \binom{n-1}{k} && \text{the next line is the binomial theorem} \\
&= n(2^{n-1} - 1) = \Omega(n2^n)
\end{aligned}$$

So yeah, this was my bad. A lot of the math above I would absolutely not expect to be followable for the majority of students in this class - I'm really only putting it here to show the **work** that goes into obtaining tight lowerbounds and to partially make the case that there are good reasons to take more math classes in your time here if you're able.

(b) A **permutation** of an ordered list is just a reordering of that list. For example, $[2, 1, 3]$ is a permutation of the list $[1, 2, 3]$

i. Write out the 6 permutations of $[1, 2, 3]$

Solution: $[1, 2, 3]$, $[1, 3, 2]$, $[2, 1, 3]$, $[2, 3, 1]$, $[3, 1, 2]$, $[3, 2, 1]$

ii. How many permutations of a list of n items are there?
 $n!$

iii. Write out an algorithm GENERATEALLPERMUTATIONS to create a list of all the permutations of a given array. It should look pretty similar to the algorithm above for GENERATEALLSUBSETS. Analyze your algorithm and show that it has a runtime of $\Theta(n \times n!)$.

Solution:

```
def GenerateAllPermutations(A):
    output = [[]] # A list containing an empty list
    for x in A:
        newList = [] # Another new list
        for lst in output:
            # put x in every possible index in a copy of lst
            for j in range(len(lst) + 1):
                copyLst = copy(lst) # A copy of lst
                copyLst.insert(j, x)
                newList.append(copyLst)
        output = newList
    return output
```

```

        add x at index j of copyLst
        add copyLst to newList
    output = newList
    return output

```

For this one, the lowerbound is actually the easier side. There are $n!$ permutations, each of which is of length n , so even to just write down that list will take $\Omega(n \times n!)$ time.

To get the upperbound of the runtime, note that at the start of iteration i of the outer for loop the size of `output` is $i!$, as it contains all permutations of the first i items of A . During this specific iteration, it iterates over all the lists in `output` (each of which is of size i), and enters into a new for loop of length $i + 1$, where each iteration of this inner for loop takes $\Theta(i)$ time. This gives us a sum like

$$\begin{aligned}
 \sum_{i=0}^{n-1} i! * i^2 &< \sum_{i=0}^{n-1} i! n^2 && \text{since } i < n \\
 &= n^2 \sum_{i=0}^{n-1} i! \\
 &\leq n^2 \sum_{i=0}^{n-1} \frac{(n-1)!}{2^i} && \text{explanation below} \\
 &= n^2 (n-1)! \sum_{i=0}^{n-1} \frac{1}{2^i} \\
 &= n \times n! \sum_{i=0}^{n-1} \frac{1}{2^i} \\
 &\leq n \times n! \times 2 && \text{assignment 1} \\
 &= O(n \times n!)
 \end{aligned}$$

For that line that says explanation below, all I'm doing is pointing out that since $i \leq n - 1$ and $(n - 1) \times (n - 2) \times \dots \times (n - i) > 2^i$ (for reasonably big n - this was from assignment 1 actually), we get that $i! \leq \frac{(n-1)!}{2^i}$

Note: No more solutions to follow because next week's assignment is to pick more of these exchange argument problems. Next week I'll ask you all to pick 5 for me to write out the solutions for :)