Homework 2: Book Work

Alex Petralia

February 2019

1 Homework 2: Book Work

Question 1: part a

Let $A = \{R, G, B, O, Y\}$ where R = Red, G = Green, etc.
We note that the total number of arrangements of picking 3 random cards after the deck is shuffled will be a permutation since the ordering matters. We want to calculate this to see how many different pairs of 3 are possible in total.

To calculate the number of permutations the formula is $\frac{n!}{(n-r)!}$
where $n = 5$ (size of the set) and $r = 3$ (sample size).
Therefore $\frac{n!}{(n-r)!} = \frac{5!}{(5-3)!} = 60$ total permutations.

Let $B = \{B, G, O, R, Y\}$ where set B is just set A but in alphabetical order. In order to calculate the total number of alphabetical pairs of 3 cards we must find the total number of combinations since the order does not matter in this case as the set is already in alphabetical order.
The formula is $\frac{n!}{r!(n-r)!}$ where $n = 5$ and $r = 3$.
Thus $\frac{n!}{r!(n-r)!} = \frac{5!}{3!(5-3)!} = 10$ total combinations.

We then divide the total combinations by the total permutations to get the probability which gives us $\frac{10}{60} = \frac{1}{6}$ for the probability.

Question 1: part b

In order to calculate the probability we will take a similar approach from Question 1.1 but this time we will set use k as our r value.
Let $C = \{1, 2, 3, 4, ..., n\}$ and $r = k \leq n$.
We will now setup our total permutations as: $\frac{n!}{(n-k)!}$.
Noting that we use k instead of r where $k \leq n$.
For total combinations of pairs of size k we use: $\frac{n!}{k!(n-k)!}$.

Question 1: part b Continued

Now we will divide the total combinations by the total permutations to reach a function that produces the wanted probability.
Thus our probability is $\frac{(\frac{n!}{k!(n-k)!})}{(\frac{n!}{(n-k)!})} = \frac{n!}{k!(n-k)!} * \frac{(n-k)!}{n!} = \frac{1}{k!}$.
Therefore our probability is $\frac{1}{k!}$.

Question 2

First let's list a hierarchy for the class of growth functions as follows from slowest to fastest growth rate: $1, log(n), \sqrt{n}, n, nlog(n), n^2, n^3, 2^n, n!, n^n, 2^{2n}, ...$

Now we will look at all the *log* functions as these are known to have a slow growth rate.
We have $(\sqrt{2})^{log(n)}, log^2(n), n^{log(log(n))}, \sqrt{log_2(n)}, 2^{log(n)}, \sqrt{log_2 n}$
which are functions $f_1(n), f_3(n), f_4(n), f_6(n), f_9(n)$ in that order.

Knowing the identity $x^{log_b(y)} = y^{log_b(x)}$ we can simplify $f_1(n), f_4(n)$, and $f_6(n)$.
Therefore $f_1(n) = n^{log(\sqrt{2})}, f4(n) = log^{log(n)}(n), f_6(n) = n^{log(2)}$.
From this we can deduce that the hierarchy of the *log* functions is:
$f_9(n) < f_1(n) < f_3(n) < f_6(n) < f_4(n)$.

Now we are left with $f_2(n) = n^2, f_5(n) = (n+1)!, f_7(n) = 1, f_8(n) = (\frac{3}{2})^n$.
$f_7(n)$ will be the slowest growth rate in our whole hierarchy.
Looking at $f_2(n)$ and $f_8(n)$ we note that $n^2 < n^3 < 2^n$ which is enough to conclude that $f_2(n) > f_8(n)$.
Lastly $f_5(n)$ is the fastest growth rate as it is a factorial function.

Thus we have enough to conclude our hierarchy (slowest to fastest growth rate):

$f_7(n) < f_9(n) < f_1(n) < f_3(n) < f_6(n) < f_4(n) < f_8(n) < f_2(n) < f_5(n)$.

Question 3: part a

For the pseudocode we first want to set the length and a very small number (whatever the programming software allows) so that we can have any new sum replace it in the first iteration even if it's negative or 0.
Then we need to use 2 for-loops in order to have two indices for our sums.

```
MAX-SUBARRAY(int[] A){
    int n = A.length
    int sum = (extremely small negative value)
    for(i = 0 to n){
        int sum = 0;
        for(int j = i to n){
            sum += A[j];
            if(sum > max)
                max = sum;
        }
    }
    return max;
}
```

Question3 : part b

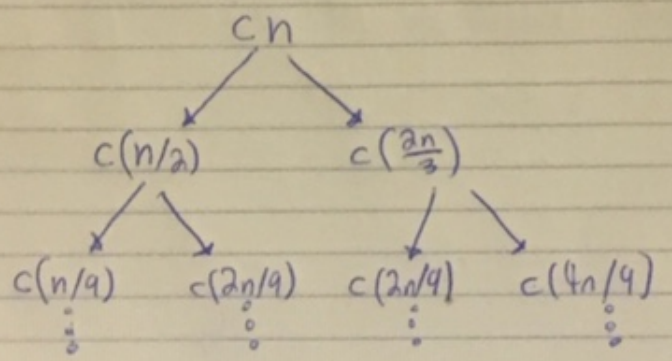
At every iteration of i we are iterating just as much when iterating through j since we are tallying up all the sub arrays to see if it beats the max value which is our end-result sum.
Since i and j iterate n-times for each of them we will end up with
$\Theta(n * n) = \Theta(n^2)$.

Question4

★ Question 4 ★

$T(n) = T(\frac{n}{3}) + T(\frac{2n}{3}) + cn$

Recursive Tree (with cn)



The shortest path is to the left and being divided by 3.

Thus the total length of the left-most path is $\log_3 n$.

Consequently the cost of this path will equals $T(n) = cn\log_3 n$.

If we convert the T(n) we found in Big Omega notation we get $T(n) = cn\log_3 n = \Omega(n \log n)$.