Student name: Quan Hong Doan

Student ID: 986956

Assignment: Lab 3

Week: 03

Due: Nov 04, 2019

**Lab 3**

**Problem 1**:

**Answer**:

If we have an array has n items, the search time will be less then .

If it is at 1 then we found it in 1 comparison; if it is at 2, we found it in 2 comparisons, and so on, we do 𝑛 comparison in order to find it. To average it we will sum the total number of comparisons 1 + 2 + ⋯ + 𝑛 = and divide it by 𝑛 (size of the array) resulting in .

In our case, the array has at least one A and B; also A and B are equally likely to be presented in the array. That means the n here is 2 and what will be from the average sum is .

**Problem 2**:

**Answer**:

If the input array is already-sorted array, the running time is that mean we have , c is a constant value.

If the input array is reversed-sorted array, then its running time is , similarly we will have its formula of asymptotic time , which d is another constant for this formula.

Because “only one of these arrays is ever reversed-sorted”, that means the distribution between already-sorted and reversed-sorted is and respectively.

That means:

**Problem 3**:

**Answer**:

1. The expected number of tosses required to get a 6.

The probability of rolling a 6 is 1/6 and 5/6 for other numbers.

In this case, the expected number would be:

By solving the above, we have E=6, which is the expected number of tosses required to get a 6.

1. The expected number of tosses required to get a total of three 6?

Similarly, now we need to find the repeated occurrence of a 6. Because we need to know only the total, not the consecutive number, that means each occurrence of 6 is independent to the other, therefore the expected number is a summation of each, which is 6 + 6 + 6 = 18.

**Problem 4**:

**Answer**:

The problem is a subset sum problem, and we have few choices:

* If we use the recursive formula, we will have a worst case,
* If we apply dynamic programming here, that will be O(n)

With the dynamic programming:

* Create a True/False 2D table subset[][]
* Fill the table in bottom up manner.
* The value of subset[i][j] will be True if there is a subset of set[0..j-1] with sum equal to **i**, otherwise false.
* Finally, we return subset[sum][n]

**Problem 5**:

**Answer**:

The problem is at step 2, “randomly permute”. If randomly permuting a number from array *arr*:

* We don’t know if we check the same number or not.
* By randomly permuting, the pointer also needs to move and access randomly in the memory, that will slow down the process.

The best case is when the array *arr* is sorted, and its running time will be O(n).

The worst case is when the array is reverse-sorted, and its running time will be

The average case running time would be

**Problem 6**:

**Answer**:

With this storing value approach, its time complexity is O(n), and the extra space is O(1).

This approach is same from Dynamic Programming method, but storing the previous two numbers only because we only need to find the next number in series.