# BLG335E: Analysis of Algorithms I Report #1

Uğur Ali Kaplan - 150170042

November 5, 2019

1. In the first part of the homework, we are expected to determine the expected number of duplication in a randomly generated array of length n which we will denote by $A[1...n]$. Elements of the $A[1...n]$ are uniformly randomly chosen from the range $[1, n]$.

   In the context of this question, duplication is defined as $A[i] = A[j]$ **and** $i < j$.

   First, say we want to analyze $X$ where $X$ is the number of duplication. To find the value of X, we would like to know how many duplication a given position $i$ has. $X_i$ is how many duplication we have for a given position i.

   With this in mind, we would have the equation $X = \sum_{i=1}^{n} X_i$.

   To find $X_i$, we can define a random variable $Y_k$ such that $k > i$ and k & i are indicators of position in the array $A$:

$$Y_k = \begin{cases} 1 & Y_k = X_i \\ 0 & Y_k \neq X_i \end{cases}$$

   With these in mind, let's continue with our analysis of X.

$$E[X] = E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] \tag{1}$$

$$E[X_i] = E[\sum_{k=i+1}^{n} Y_k] = \sum_{k=i+1}^{n} E[Y_k] \tag{2}$$

   Then, we have to find what $Y_k$ and plug the value we have found into equation 1.

   Since numbers are uniformly randomly generated, probability of getting a specific number in a given position is $1/n$. If we want to have the same value in position $i$ to appear in another position $k$, probability of that happening would be $1/n$. For position $k + 1$, it would be $1/n$ and it goes on like this until $k = n$ and there are no more positions.

   Then, using the following equations we can find the expected value of $X_i$.

$$X_i = Y_k + Y_{k+1} + ... + Y_n \tag{3}$$

   For $x \in [i + 1, ..., n]$, $E[Y_x] = 1/n$.

$$E[X_i] = E[Y_k] + E[Y_{k+1}] + ... + E[Y_n] \tag{4}$$

$$E[X_i] = 1/n + 1/n + ... + 1/n \tag{5}$$

Number of $1/n$'s in this equation is $n - (i + 1) + 1 = n - i$. Thus,

$$E[X_i] = (n - i)/n \tag{6}$$

With this, we can plug this value into equation 1 like we said we would and then we have

$$E[X] = \sum_{i=1}^{n} (n - i)/n = ((n - 1) + (n - 2) + ... + 1 + 0)/n = (n - 1)/2 \tag{7}$$

So, for a given $n$, expected number of duplication is $(n - 1)/2$.

If we compare the expected number of duplication and the results in our homework, we get this:

| n | Expected | Program Result |
|---|---|---|
| 30 | 14.5 | 14 |
| 100 | 49.5 | 48 |

2. In this second part of the homework, we have to calculate the running time of our program.

   Since constant running time of operations does not matter in asymptotic analysis, instead of giving each line its own running time(e.g. line 1 has $c_1$, line 2 has $c_2$ and so on), I will group this constants into a big constant variable $\mathcal{C}$.

   Our program starts by taking an argument $n$ from the command line which will be the number of elements array $\mathcal{A}$ contains and also the maximum value an element in $\mathcal{A}$ can take.

```
1    unsigned int arr_length; // User will provide the array length and maximum number we
     can generate
2    arr_length = stoul(argv[1]);
3    unsigned int *random_arr = new unsigned int[arr_length];
```

Listing 1: Snippet 1

   Both of 3 lines seen in this snippet run once and they have nothing to do with the given input $n$.

   Then, we continue with creation of an array, that has $n$ elements and each element has a value between 1 and $n$.

```
1    unsigned seed = chrono::system_clock::now().time_since_epoch().count();
2    default_random_engine generator(seed);
3    uniform_int_distribution<int> distribution(1, arr_length);
4
5    for (unsigned int i = 0; i < arr_length; i++) {
6        random_arr[i] = distribution(generator); // Element i is between 1 and
     arr_length(included)
7    }
```

Listing 2: Snippet 2

   Again, first 3 lines in this code snippet run only once. Likewise, variable declaration for the for loop runs only once as well. Let's group running time we have so far into a big constant $\mathcal{C}_1$.

   Loop condition(Running time $= \mathcal{C}_2$) will run $(n+1)$ times, increment operation and code inside the for loop(Running time $= \mathcal{C}_3$) will run $n$ times.

   We can say our running time so far is $\mathcal{C}_1 + (n) \times \mathcal{C}_3 + (n+1) \times \mathcal{C}_2$. But to have a simpler expression for the running time we can do the following:

   $(n+1) \times \mathcal{C}_2$ can be written as $n \times \mathcal{C}_2 + \mathcal{C}_2$.
   $\mathcal{C}_2$ can be grouped into $\mathcal{C}_1$ and $n \times \mathcal{C}_2$ can be grouped into $n \times \mathcal{C}_3$.

   We can say our running time so far is $\mathcal{C}_1 + (n) \times \mathcal{C}_3$

Now, we want to print elements of the array.

```
1    cout << "ARRAY:";
2    for (unsigned int i = 0; i < arr_length; i++) {
3        if (i % 10 == 0) cout << "\n"; // 10 elements per line
4        cout << random_arr[i] << "\t";
5    }
6    cout << endl;
```

Listing 3: Snippet 3

Lines 1 and 6 will run only once, as well as the variable declaration on line 2. Loop condition will run $(n+1)$ times. Increment operation and the contents of the loop which prints onto screen will run $n$ times. Asymptotically speaking, nothing has changed so far. We have $\mathcal{C}_1 + (n) \times \mathcal{C}_3$.

```
1    cout << "\nEXPECTED NUMBER OF DUPLICATIONS = " << (arr_length - 1) / 2.0 << endl;
2
3    vector<unsigned int> dup_vec;
4    find_duplications(random_arr, arr_length, dup_vec);
5
6    int c = 0;
7    for (auto i = dup_vec.begin(); i != dup_vec.end(); advance(i, 2)) {
8        if (c % 10 == 0) cout << "\n"; // 10 elements per line
9        cout << "(" << *i + 1 << ", " << *(i + 1) + 1 << ") ";
10       c++;
11   }
```

Listing 4: Snippet 4

Line 1 in the above snippet runs once, as well as the $3^{rd}$, $6^{th}$ lines and the variable declaration on line 7.

Loop condition runs as many times as we have duplication + 1. At best, it will run 1 times since there will be no duplication. At worst, all of the numbers will be the same and it will run $(n \times (n+1)/2) + 1$ times. Increment operation and contents of the loop will again, depending on the number of duplications, run 0 times or $(n \times (n+1)/2)$ times.

So, depending on the found duplications, there are two possible total running time so far:

Best Case: $\mathcal{C}_1 + (n) \times \mathcal{C}_3$
Worst Case: $\mathcal{C}_1 + (n) \times \mathcal{C}_3 + (n^2) \times \mathcal{C}_4$

On line 4, there is a function call to find_duplications. Let's examine the contents of the function.

```
1    for (unsigned int i = 0; i < arr_length; i++) {
2        for (unsigned int j = i + 1; j < arr_length; j++) {
3            if (arr[i] == arr[j]) {
4                dup_vec.push_back(i);
```

```
                dup_vec.push_back(j);
            }
        }
    }
```

Listing 5: found_duplications

Inner loop will run n times. Outer loop will again, run n times. So, contents of the inner loop will run $n^2$ times in total. Loop condition will run $(n^2 + 1)$ times. So, since there is a call to function found_duplications in snippet 4, no matter the number of duplications found, we have a running time of $\mathcal{C}_1 + (n) \times \mathcal{C}_3 + (n^2) \times \mathcal{C}_4$.

```
    // (f)
    cout << endl << "\nENCOUNTERED NUMBER OF DUPLICATIONS = " << dup_vec.size() / 2 <<
    endl;

    delete[] random_arr;

    return 0;
```

Listing 6: Snippet 5

Each line in this snippet will only run once and will have no contributions to run time asymptotically speaking. Therefore, we have a total running time of

$$\mathcal{C}_1 + (n) \times \mathcal{C}_3 + (n^2) \times \mathcal{C}_4 \tag{8}$$

As $n \to \infty$, $n$ will be the most important factor. Since in best case and worst case for the found number of duplication, we have to call find_duplication function, we can conclude that our program has $\Theta(n^2)$ running time.