Project 1

Mengxiang Jiang
CSEN 5336 Analysis of Algorithms

September 16, 2022

Problem 1. a) Choose a number N between 5,000,000 and 10,000,000. Generate an array of 1,000,000 random numbers within the range of 1 to N.
[Hint: Code to generate the numbers in python is given below]

```
import random
randmatrix = [0]*1000000
for i in range(1000000):
    randmatrix[i] = random.randint(1, N)
```

Listing 1: Generate Array with Random Integers

b) Find a simple and efficient algorithm to find the median of the array. Explain why your algorithm is simple and efficient compared with some other algorithms.

Ok, so reading the Python documentation about the *randint* function, it states that it uses the *randrange* function, which produces equally distributed values across the given range. This means we can use bucket sort to sort the *randmatrix* in O(n) time, and then return the middle element(s) of the array as the median.

This algorithm is efficient since the running time is O(n) for the bucket sort, and O(1) for the middle element access, so the total running time is O(n), which is as fast as the most efficient selection algorithm for unsorted data. Bucket sort is also better suited for this than counting sort, since the length of the array is less than the range of the integers, so buckets are better utilized than counts (which will mostly be 0s).

This algorithm is also simpler than the Blum, Floyd, Pratt, Rivest, Tarjan Algorithm. This is because although both algorithms are O(n), bucket sort is very intuitive (at least conceptually if not implementation wise), while the Blum, Floyd, Pratt, Rivest, Tarjan algorithm has seemingly arbitrary constants and much harder to understand.

c) Submit your code to solve the problem.

```
def insertionSort(A):
    for i in range(1, len(A)):
        j = i
        while j > 0 and A[j - 1] > A[j]:
            A[j], A[j - 1] = A[j - 1], A[j]
            j -= 1
    return A

def bucketSort(A, A_min, A_max):
    A_range = A_max - A_min
    A_len = len(A)
    B = [[] for i in range(A_len)]
    for a in A:
        i = int(A_len * (a - A_min) / A_range)
        # Since bucket sort assumes interval is [0, 1)
        # but randint(1, N) doesn't exclude N
        if i == A_len:
            i -= 1
        B[i].append(a)
    for b in B:
        insertionSort(b)
    C = [c for b in B for c in b]
    return C

def median(A, A_min, A_max):
    C = bucketSort(A, A_min, A_max)
    if len(A) % 2 == 1:
        return C[len(A) // 2]
    else:
        return (C[len(A) // 2] + C[len(A) // 2 - 1]) / 2
```

Listing 2: Find Median of a Random Array

```
>>> import random
>>>
>>> N = 5000000
>>> randmatrix = [0]*1000000
>>> for i in range(1000000):
...     randmatrix[i] = random.randint(1, N)
...
>>> m = median(randmatrix, 1, N)
>>> m
2497575.0
>>> import statistics
>>> m2 = statistics.median(randmatrix)
>>> m2
2497575.0
```

Listing 3: Sample execution of function in terminal