19AIE203-Data Structures and Algorithms 2

Project Report

Done by

Name: M Mahadev **Name: Marasani Jayasurya**

Roll No: AM.EN.U4AIE20045 **Roll No: AM.EN.U4AIE20048**

Randomized Algorithms

Introduction:

Randomized algorithms are used when presented with a time or memory constraint, and an average case solution is an acceptable output. Due to the potential erroneous output of the algorithm, an algorithm known as amplification is used in order to boost the probability of correctness by sacrificing runtime. Amplification works by repeating the randomized algorithm several times with different random subsamples of the input and comparing their results. It is common for randomized algorithms to amplify just parts of the process, as too much amplification may increase the running time beyond the given constraints.

Randomized algorithms are usually designed in one of two common forms: as a Las Vegas or as a Monte Carlo algorithm. A Las Vegas algorithm runs within a specified amount of time. If it finds a solution within that timeframe, the solution will be exactly correct. however, it is possible that it runs out of time and does not find any solutions. On the other hand, a Monte Carlo algorithm is a probabilistic algorithm which, depending on the input, has a slight probability of producing an incorrect result or failing to produce a result altogether.

Implemented Algorithms:

1. **Approximation of π**

   The idea is to simulate random (x, y) points in a 2-D plane with domain as a square of side 1 unit. Imagine a circle inside the same domain with the same diameter and inscribed into the square. We then calculate the ratio of number points that lie inside the circle and total number of generated points.

   Algorithm:

   radius= 1
   At the start, both points inside circle and square are 0
   we then randomly generate (x and y) range: (-1 to 1)
   The range (-1 to 1)
   if (x, y) lies inside the circle:
   if it lies inside the circle:
   increment the circle points and square points

if not:
increment only the square points
pi = 4 * circle_points /square_points
print(pi)

Output Screenshot:

```
Enter the number of iteration: 100
Final Estimation of Pi= 3.1444
```

2. **Randomized Binary Search**

In randomized binary search, instead of picking the middle element, we pick a random element within the range as our pivot. That's the only difference and the rest of the things are the same. Randomized binary search has logarithmic time complexity i.e O(Log(n))

Algorithm:

Pivot= random index in between [left, right]
which can be found like below:
pivot=left+ rand () %(right-left+1)
If array[pivot]==key
Key is found. Return
Else If array[pivot]<key
Search only in the right half,
thus, set left= pivot +1, new range [pivot+1, right]
Else if array[pivot]>key
Search only in the left half,
thus, set right= pivot -1, new range [left, pivot-1]
If left>right
Break and element is not found

Output Screenshot:

```
Enter the elements of the array: 8 7 4 5 3 6
Enter the element you need to search: 4
The searched element is not present
```

3. **Randomized Quick Sort**

Randomized Quick Sort, we use a random number to pick the next pivot (or we randomly shuffle the array). In QuickSort using random pivoting we first partition the array in place such that all elements to the left of the pivot element are smaller, while all elements to the right of the pivot are greater than the pivot.

Algorithm:

1) The function random generator generates random integer:
between the starting index and the ending index
2) The function partitioner is to swap the starting index of the array and the pivot
3) The function partition is to rearrange according to the pivot
Element < pivot: to the left of the pivot
Element > pivot: Placed to the right of the pivot

4) The function quicksort:
if stat index < stop index:
we partition it and then:
and call left quick sort (left of the pivot)
and right quick sort (right of the pivot)

Output Screenshot:

```
Enter the elements of the array:  34 2 42  2 2
[2, 2, 2, 34, 42]
```

4. **Approximate Median Algorithm**

This algorithm is a non-deterministic algorithm which takes O((Log n) x (Log Log n)) time and produces incorrect results with probability less than or equal to $2/n^2$.

Algorithm:

- Randomly choose k elements from the array where k=c log n (c is some constant)
- then, Insert into a set.
- Sort elements of the set.
- Return median of the set i.e. $(k/2)^{th}$ element from the set

Time Complexity: O (log n (log log n))

The algorithm produces incorrect results with probability less than or equal to $2/n^2$.

Input Screenshot:

```
Enter the size of array: 10
Enter the values of array:
1
2
3
4
5
6
7
8
9
0
```

Output Screenshot:

```
Enter the any contstant value1000
Approximate Median: 4
```

5. **Freivald's Algorithm for Matrix Product Checking**

Freivalds Algorithm probabilistic randomized algorithm used to verify matrix multiplication. Given three n x n matrices, Freivalds' algorithm determines in $o(kn^2)$ whether the matrices are equal for a chosen k value with probability of failure less than $1/2^k$.

Algorithm:

- Input n*n Matrices, i.e A, B and C.
- To verify A×B=C, choose a n×1 column vector **r** with randomly choosing 0 or 1.
- Compute A×(B×r) and C×r which takes $O(n^2)$ time.
- if A×(B×r) ≠ C×r, Then Output False else if A×(B×r) = C×r Output True.

Complexity

- Worst case time complexity: $\Theta(kn^2)$
- Space complexity: $\Theta(n^2)$

where k = number of times the algorithm iterates.

Error Analysis

Probability of Error,

Case 1: A×B = C The Probability of Error is 0

Case 2: A×B ≠ C The Probability of Error is less than 1/2 and for k iterations probability of failure less than $1/2^k$.

Output Screenshot:

```
Enter the number of iterations:
100

False
```

6. **Primality Testing**
   - **Fermat Method**

     The Time Complexity for this test is O(K Log n)

     Fermat's Little Theorem: If n is a prime number, then for every a, 1 < a < n-1,a^n(n-1) ≡ 1 (mod n) OR a^(n-1) % n = 1

     If a given number is prime, then this method always returns true. If the given number is composite (or non-prime), then it may return true or false, but the probability of producing incorrect results for composite is low and can be reduced by doing more iterations.

     Output Screenshot:

     ```
     Enter the Number:34059687
     Number of Iterations:400
     The number is not Prime
     ```

   - **Solovay Strassen**

     Solovay Strassen

     We divide Solovay Strassen Primality Test algorithm in following two parts:

     - Find the value of Euler Criterion formula
     - Find Jacobi Symbol for given value

     1. Euler Criterion Formula

     $$a^{\frac{n-1}{2}} \equiv x \ (mod\ n)$$

     Where,

     a: any random variable from 2 to (n-1).

     n: given number for primality test.

     2. Jacobi Symbol

     Algorithm: -

```
Jacobi (a, n) {
    j = 1
    while (a not 0) do {
        while (a even) do {
            a = a/2
            if (n = 3 (mod 8) or n = 5 (mod 8)) then
                j = -j
            }
```

```
        interchange (a, n)
        if (a = 3 (mod 4) and n = 3 (mod 4)) then
            j = -j
            a = a mod n
    }
    Return j
}
```

When a is even positive number then

$$\left(\frac{a}{n}\right) = (-1)^{\left(\frac{n^2-1}{8}\right)} = \begin{cases} 1, & if\ n \equiv 1,7 (mod\ 8) \\ -1, & if\ n \equiv 3,5 (mod\ 8) \end{cases}$$

When a is odd positive number then

$$\left(\frac{a}{n}\right) = (-1)^{\left(\frac{n-1}{2}\right)} = \begin{cases} 1, & if\ n \equiv 1 (mod\ 4) \\ -1, & if\ n \equiv 3 (mod\ 4) \end{cases}$$

We compare this Jacobi symbol with Euler criterion formula and if both are same then the number is Prime and if both are different then number is Composite.
The time complexity of this test is O(k Log n).It is possible for the algorithm to return an incorrect answer. If the input n is indeed prime, then the output will always probably be correctly prime. However, if the input n is composite, then it is possible for the output to probably be an incorrect prime. The number n is then called an Euler-Jacobi pseudoprime.

Output Screenshot:

```
Enter the number: 3405987
Enter the number of iterations: 500
The number is not Prime
```

Advantages of Randomized Algorithms:

- Speed of a randomized algorithm may be faster than any deterministic algorithm.
- These algorithms are simpler even if not faster.
- Sometimes randomized algorithms are best for practical scenarios.
- Randomized ideas lead to deterministic algorithms.