CAB301 Practical 2

Time: 11:00 Monday

**Question 1**: Definitions of big-O notation

a) When the time **grows in proportion** to the size, the complexity is of class

b) Time does not change, regardless of input size, the complexity is of class

c) . Since the highest power is , the complexity is of class

**Question 2**: Find efficiency function and conclude the time complexity in big- notation.

For all parts:

* The parameter characterising the input size is the number itself, .
* The basic operation is .

Part a)

**for** **to** **do**

Assume the time taken to run each basic operation is .

In this for loop, the basic operation is executed for every number from to , so the summation representing this for loop is:

\* **Explanation**: Since there are numbers between and , the number inside the summation is added to itself times.

Since the highest power multiplying (the time of the basic operation) is , we say the time complexity of the algorithm is .

c)

**if**   
 **for** **to** **do**

**else**  
 **for** **to** **do** **for** **to** **do**

Scenario, in a lottery, if you win, you get $, but if you don’t, you lose $ because you need to buy a ticket.

Let’s say there is a 50% chance of winning.

In two rounds, you win once, and lose once. The net income is: . On average, the income for each round is:

If the probability is , so intuitively we think that for every 10 tickets, we would win once:

* 9 loses:
* 1 win:

Total: , so on average, the income per ticket is:

If we don’t know the probability, call it , where , we can find the average income per ticket:

Apply to our case:

Start by finding the complexity of each case separately.

When :

When :

Take into account the probability of . Set this probability to , where .

The final efficiency function:

**Question 3**: Analysis of Algorithm. Only need the big- notation (even without an efficiency function).

Part b) Algorithm to find the minimum separation between any two elements in an array

**ALGORITHM** MinDistance()

**for** **to** **do**  
 **for** **to** **do**

**return**

The parameter characterising the input size is the number of elements in the array , which is called .

The basic operation chosen is the assignment .

Assume the time taken for one basic operation is . We can set up the summation formula to represent the time taken for the whole algorithm to run:

For the inner summation:

There are numbers between and :

Therefore:

And the complexity function becomes:

Split the sum to two parts, one dependent on , and one independent of :

The next step is to work out

Add the above together:

Here, is added to itself times, therefore:

Divide both sides by 2 to get the summation:

Putting all back together, the complexity is:

After the above proof, where the worst-case time complexity is , we find the highest power multiply is . Therefore, complexity is of class .

**Question 5:**

namespace SortAnalysis

{

internal class Program

{

static void Main(string[] args)

{

Console.WriteLine("Size\t,Count");

for (int n = 1000; n <= 20000; n += 1000)

{

int count = RunAlgorithm(n);

Console.WriteLine($"{n}\t,{count}");

}

}

/// <summary>

/// Run the algorithm on a random array of n

/// elements

/// </summary>

/// <param name="n">The number of elements in the array</param>

/// <returns>The count of the basic operations</returns>

static int RunAlgorithm(int n)

{

// Create a random array of size n

Random rnd = new Random();

int[] arr = new int[n];

for (int i = 0; i < n; i++)

{

arr[i] = rnd.Next();

}

return SortAnalysis(arr);

}

static int SortAnalysis(int[] arr)

{

int n = arr.Length;

int count = 0;

for (int i = 1; i <= n - 1; i++)

{

int v = arr[i];

int j = i - 1;

// If statement added to simulate

// A[j] > v when j < 0

if (j < 0) count++;

while (j >= 0 && arr[j] > v)

{

count++;

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = v;

}

return count;

}

}

}

|  |  |
| --- | --- |
| **Size** | **Count** |
| 1000 | 257751 |
| 2000 | 970634 |
| 3000 | 2258420 |
| 4000 | 4036676 |
| 5000 | 6183260 |
| 6000 | 9013021 |
| 7000 | 12324177 |
| 8000 | 16090174 |
| 9000 | 20158871 |
| 10000 | 24687701 |
| 11000 | 30008901 |
| 12000 | 35917849 |
| 13000 | 42033920 |
| 14000 | 48614915 |
| 15000 | 55979051 |
| 16000 | 64356264 |
| 17000 | 72109924 |
| 18000 | 80798052 |
| 19000 | 90382533 |
| 20000 | 99646226 |

After collecting data, form a hypothesis. If the count is a function of size, i.e.:

Try:

If , then

If , then

In our case, take :

Try more if needed.

Find that as the number of input doubles, the time taken quadruples, hence the complexity is .

Plotting count against size also shows a quadratic relationship:

