Data structures and algorithms

Tutorial 1

Amr Keleg

Faculty of Engineering, Ain Shams University

February 29, 2020

Contact: amr_mohamed@live.com

Outline

Outline

- How to analyze the running time of a code snippet?
- There are many factors affecting the running time:
  - The specifications of the running computer
  - The programming language
  - The compiler
  - ....
- We don't need to find the exact running time. We just want to know how will the running time scale.

- Express the running time in terms of the size of the input (size of an array,length of string, value of integer, ...)
- Worst case analysis

Example:

```
bool is_element_in_array(int arr[], int element, int arr_size){
  for (int i=0; i< arr_size; i++)
  {
    if (arr[i] == element)
    {
      // The element exists in the array
      return true;
    }
  }
  return false;
}
```

Outline

Question 1 - a

```
for (int i=0; i< N; i++)
  cout<<i;
```

Instruction	# of operations	Cost of each operation
int i=0	1	1
cout<<i;	N	1
i++	N	1
i<N	N+1	1

Total no. of operations $= 1 + N + N + (N + 1) = 3N + 2$
The running time of this code snippet is said to be in $O(N)$

Formal Definition:
f(n) is said to be in O(g(n)) if: C*g(n) $>$f(n) for all n $>$n0
where n0 (n0$>$0), C (C$>$0)

In other words, g(n) is an upperbound for the function f(n) for a sufficiently large n.

Question 1 - b

```
for (int i=1; i< N; i*=2)
   cout<<i;
```

Instruction	# of operations	Cost of each operation
int i=0	1	1
cout<<i;	$\lceil \log_2 N \rceil$	1
i++	$\lceil \log_2 N \rceil$	1
i<N	$\lceil \log_2 N \rceil + 1$	1

Total no. of operations $= 1 + \lceil \log_2 N \rceil + \lceil \log_2 N \rceil + (\lceil \log_2 N \rceil + 1) = 3\lceil \log_2 N \rceil + 2$
The running time of this code snippet is said to be in $O(log_2 N)$ or $O(\log N)$

Before digging deeper, Let's sort the complexities in an ascending order:

1. O(1)
2. O(N)
3. $O(N^2)$
4. $O(2^N)$
5. $O(\sqrt{N})$
6. $O(N * \log N)$
7. $O(\log N)$
8. $O(N!)$

Question 1 - c

```
for (int i=0; i< N; i++)
   f(n); // O(log(N))
```

Instruction	# of operations	Cost of each operation
int i=0	1	1
f(n);	N	log(N)
i++	N	1
i<N	N+1	1

Total no. of operations = 1 + N*log(N) + N + (N + 1) = N*log(N) + 2N + 2

The running time of this code snippet is said to be in O(N*log(N))

Outline

Outline

1. Big-OH notation O(.)
   - Definition
   - Sheet 1 - Question 1
2. Recursive Functions
   - Sheet 1 - Question 1 - d
   - How to write Recursive Functions?
   - Master Method and Recursion Trees
   - Sheet 1 - Question 1d/7b

```
void decimal2binary (int n)
{
  if(n>0)
  {
    decimal2binary(n/2);
    cout<<n%2;
  }
}
```

What's the idea of the code?

What's its complexity?

Outline

- Write the base case (the case where the input is trivial).
- Make the function call itself again.
- Make sure the input to the recursive call is decreasing (getting closer to the base case).

Sheet(1) - Question (3):
Write a program that recursively finds the maximum value of an array of integers.

Inputs to the function: The array and its length.
Output: The maximum value of the array.

Sheet(1) - Question (3):
Write a program that recursively finds the maximum value of an array of integers.

Base Case: An array of length 1
Recursive call: ??

```
int find_max(int arr[], int length){
  if (length==1)
    return arr[0];
  return max(arr[length-1], find_max(arr, length-1));
}
```

More ideas to practice recursion:

- Find the sum of the elements in an array.
- Print all the even numbers in range [0, N] where N is an even number.
- Compute the value of factorial of N.
- Compute the value of the Nth Fibonnaci coefficient (fib(N)). fib(N) = fib(N-1) + fib(N-2), fib(0) = 0, fib(1) = 1.

Sheet(1) - Question (2):
Create a C/C++ function that takes an array of integers and returns whether it has repeated items or not. Determine the big O of your solution. Repeat if the array is sorted.

```
bool contains_repeated(int arr[], int N);
```

```
bool contains_repeated(int arr[], int N){
  for (int i=0; i<N; i++)
  {
    for(int j=0; j<N; j++)
    {
      if (i != j && arr[i] == arr[j])
        return 1;
    }
  }
  return 0;
}
```

Complexity?

```
bool contains_repeated(int arr[], int N){
  for (int i=0; i<N-1; i++)
  {
    for(int j=i+1; j<N; j++)
    {
      if (arr[i] == arr[j])
        return 1;
    }
  }
  return 0;
}
```

Complexity?

```
bool contains_repeated(int arr[], int N){
  for (int i=0; i<N-1; i++)
  {
    if (arr[i] == arr[i+1])
      return 1;
  }
  return 0;
}
```

Complexity?

Outline

1. Big-OH notation O(.)
   - Definition
   - Sheet 1 - Question 1
2. Recursive Functions
   - Sheet 1 - Question 1 - d
   - How to write Recursive Functions?
   - Master Method and Recursion Trees
   - Sheet 1 - Question 1d/7b

The Master Method

If $T(n) \leq aT\left(\frac{n}{b}\right) + O(n^d)$

then

$$T(n) = \begin{cases} O\left(n^d \log n\right) & \text{if } a = b^d \quad \text{(Case 1)} \\ O(n^d) & \text{if } a < b^d \quad \text{(Case 2)} \\ O(n^{\log_b a}) & \text{if } a > b^d \quad \text{(Case 3)} \end{cases}$$

Outline

1 Big-OH notation O(.)
- Definition
- Sheet 1 - Question 1

2 Recursive Functions
- Sheet 1 - Question 1 - d
- How to write Recursive Functions?
- Master Method and Recursion Trees
- Sheet 1 - Question 1d/7b

How to express the running time in terms of a recurrence relation:

```
void decimal2binary (int n)
{
  if(n>0)
  {
    decimal2binary(n/2);
    cout<<n%2;
  }
}
```

$T(n) = ??$

$T(n) = T(n/2) + 2$

$a = 1$ $b = 2$ $d = 0$

$T(n)$ is $O(\log(n))$

Feedback form:

Amr: `https://forms.gle/dWibC11k95m4MK4H9`

Fady: `https://forms.gle/ooYqrgeGRyrk8sXk7`