Lecture 1: Intro To Time Complexity and Arrays

Class	Cohort 2 Year 1
Created	@Sep 30, 2020 4:23 PM
Materials	
Reviewed	☐
Type	Lecture

Introduction

- What are algorithms?
  - What do we mean by a valid algorithm?
  - Why is there a need for comparing algorithms?
- What are the important criteria for comparing algorithms?
  - Time and Space
- What are the problems of experimental comparison?
  - We need to have same problem size for both the algorithms that are compared.
  - We need to come up with lots of test cases to compare properly. We can't just compare based on one data point.
  - The algorithms should be implemented in the same language.
  - The algorithms should run on the same hardware.
- How do we deal with the problems of experimental comparison?

  We do analytical comparison
- What are the tools for analytical comparison?
  - Worst Case time complexity.
  - Average Case

- Best Case

```
#include <iostream>
using namespace std;

int main() {
  int x = 0;
  x += 1;
  cout << x;
  return x;
}.
```

▼ Can we count the number of operations in the above program program?

1 + 3 + 2 + 2

```
#include <iostream>
using namespace std;
int main(){
  int n = 10;
  int sum = 0;
  for (int i = 0; i < n; i++)
    sum += 2;
  cout << sum;
  return 0;
}
```

▼ What about this one?

9n+9

```
int main(){
  int n = 5;
  int m = 7;
  int sum = 0;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++)
      sum += 1;
  }
  cout << sum;
  return 0;
}
```

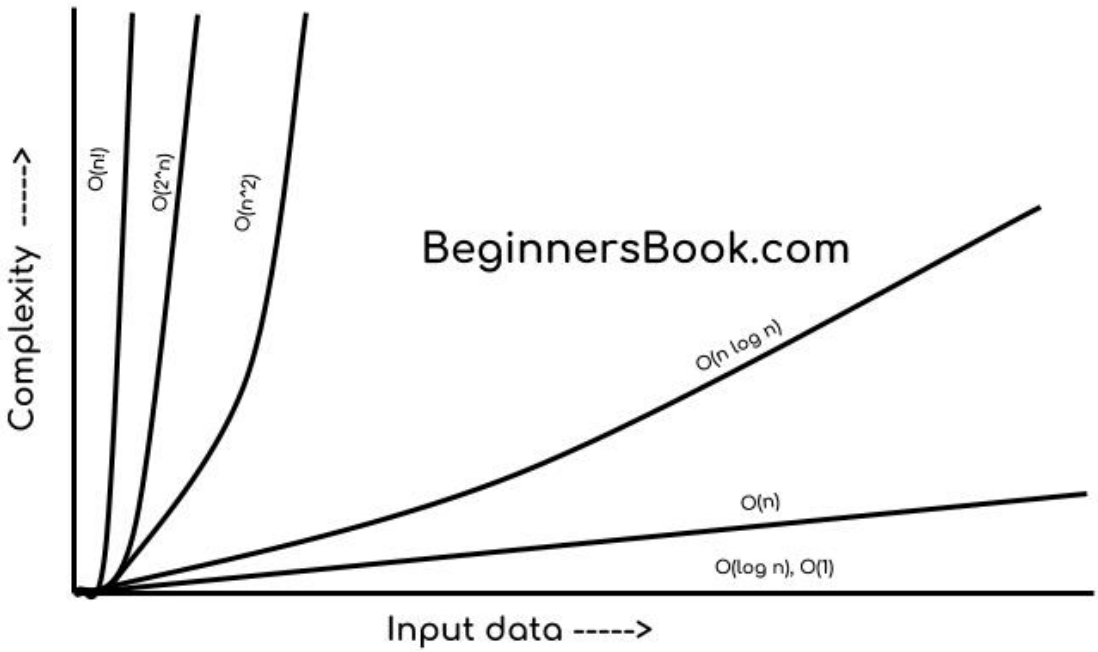
▼ What if the loops are nested?

9nm + 10n + 10

▼ What if the inner loop runs for n times?

9n^2 + 10n + 10

Asymptotic Analysis and Big O



▼ Why Big O?

Because we do not want to count again and again.

▼ What is Big O?

A function $f(n)$ is considered $O(g(n))$, if there exists some positive real constant c, and an integer $n_0 > 0$, such that the following inequality holds for all $n \geq n_0$:

$$f(n) \leq cg(n)$$

▼ Also, $O(g(n))$ is a class of functions.

▼ What are the rules for simplifying functions?

- Drop the multiplicative constants with all terms
- Drop all but the highest order term

▼ Other Notations

▼ Big "Theta" $\Theta(.)$

$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

- ▼ Little 'o' $o(.)$

  $$f(n) < cg(n)$$

- ▼ Little 'omega' $\omega(.)$

  $$f(n) > cg(n)$$

- ▼ Why Big 'O' is preferred over other notations?
  - We want the worst case time complexity
  - Big theta is not present in all the cases
  - Little 'o' and 'omega' need strict inequality. It is hard to get that sometimes.
  - Big 'O' is standard.

Some Useful Formula

Summation	Equation
$(\sum_{i=1}^{n} c) = c + c + c + \cdots + c$	cn
$(\sum_{i=1}^{n} i) = 1 + 2 + 3 + \cdots + n$	$\frac{n(n+1)}{2}$
$\left(\sum_{i=1}^{n} i^2\right) = 1 + 4 + 9 + \cdots + n^2$	$\frac{n(n+1)(2n+1)}{6}$
$\left(\sum_{i=0}^{n} r^i\right) = r^0 + r^1 + r^2 + \cdots + r^n$	$\frac{(r^{n+1}-1)}{r-1}$
$\sum_{i=0}^{n} 2^i = 2^0 + 2^1 + ... + 2^n$	$2^{n+1} - 1$

Logrithmtic expressions	Equivalent Expression
$log\ (a\ *\ b)$	$log\ (a) + log\ (b)$
$log\ (a\ /\ b)$	$log\ (a) - log\ (b)$
$log\ a^n$	$n\ log\ a$
$\sum_{i=1}^{n} log\ i = log\ 1 + log\ 2 + ... + log\ n = log(1.2...n)$	$log\ n!$

Common Scenarios

```
for (int x = 0; x < n; x++) {
    //statement(s) that take constant time
}
```

▼ What is the time complexity?

$O(n)$

```
for (int x = 0; x < n; x+=k) {
    //statement(s) that take constant time
}
```

```
for (int i=0; i<n; i++){
    for (int j=0; j<m; j++){
        //Statement(s) that take(s) constant time
    }
}
```

```
for (int i=0; i<n; i++){
    for (int j=0; j<i; j++){
        //Statement(s) that take(s) constant time
```

```
    }
}
```

```
for (int i=0; i<n; i++){
    i*=2;
    for (int j=0; j<i; j++){
        // Statement(s) that take(s) constant time
    }
}
```

```
i = //constant
n = //constant
k = //constant
while (i < n){
    i*=k;
    // Statement(s) that take(s) constant time
}
```

```
int main(){
  int n = 10;
  int sum = 0;
  float pie = 3.14;

  for (int i=0; i<n; i+=3){  // O(n/3)
    cout << pie << endl;        // O(n/3)
    for (int j=0; j<n; j+=2){    // O((n/3)*(n/2))
      sum += 1;        // O((n/3)*(n/2))
      cout << sum << endl;   // O((n/3)*(n/2))
    }
  }
}
```

```
int main(){
  int n = 10;   // n could be anything
  int sum = 0;
  float pie = 3.14;

  for (int i=n; i>=1; i-=3){
    cout << pie << endl;
    for (int j=n; j>=0; j--){
      sum += 1;
    }
  }
  cout << sum << endl;
}
```

```
int main() {
  int n = 10; //n can be anything
  int sum = 0;
  float pie = 3.14;
  int var = 1;

  while (var < n){
    cout << pie << endl;
    for (int j=0; j<var; j++)
      sum+=1;
    var*=2;
  }
  cout<<sum;
}
```

```
int main(){
  int n =10;    // you can change the value of n
  int sum = 0;
  int var = 1;
  float pie = 3.14;

  while (var < n){
    cout << pie << endl;
    for (int j = 1; j < n; j+=2){
      sum+=1;
    }
    var*=3;
  }
  cout << sum << endl;
}
```

Array

▼ How to declare arrays?

datatype arrayName[size]

▼ ArrayList

```
#include <iostream>
using namespace std;


class ArrayList {
    int *arr;
    int num_elements;
    int capacity;

public:
    ArrayList(int size) {
```

```
        arr = new int[size];
        num_elements = 0;
        capacity = size;
    }
    void insert(int val) {
        if(num_elements < capacity) {
            arr[num_elements]=val;
            num_elements++;
        } else {
            resize();
            arr[num_elements]=val;
            num_elements++;
        }
    }

    int getAt(int index){
       return arr[index];
    }

    void resize() {
        int* tempArr=new int[capacity*2];
        capacity*=2;

        for(int i=0; i<num_elements; i++) {
            tempArr[i]=arr[i];
        }

        delete [] arr;
        arr=tempArr;
    }

    int length() {

        return num_elements;
    }

    void print() {
        for(int i=0; i<num_elements; i++)
            cout << arr[i] << " ";
        cout << endl;
    }

};

int main() {
    ArrayList arr(1);
    cout << "Arr length : " << arr.length() << endl;
    arr.insert(1);
    arr.insert(2);
    arr.insert(3);
    arr.insert(4);
    arr.insert(5);
    arr.insert(6);
    arr.insert(7);
    arr.insert(8);
    cout <<  "Arr length : " << arr.length() << endl;
    cout << "Array : ";
    arr.print();
```

```
    cout << "Element at index 5 is " << arr.getAt(4) << endl;
}
```

▼ How to initialize a two-dimensional array?

datatype arr[x][y];

```
int arr[2][2] = {{1,2},{3,4}};
   arr[1][1] = 10;
```