Data Structure and Algorithms

Fairoz Matte

Introduction

1. Analysis of Algorithms
2. Mathematics
3. Arrays
4. Searching and Sorting
5. Strings
6. Matrix
7. Bit Magic
8. Recursion
9. Linked List
10. Stack

Introduction

11. Queue
12. Hashing
13. Trees
14. BST
15. Heap
16. Graph
17. Greedy
18. Dynamic Programming
19. Trie

Analysis of Algorithms

```
int function1(int n)
{
    return n*(n+1)/2;
}
```

n=3 (1+2+3)

0/p = 6

n = 5 (1+2+3+4+5)

o/p = 15

Analysis of Algorithms

```
int function1(int n) {
     return n*(n+1)/2;
}

int function2(int n) {
    int sum =0;
   for(int i=0; i<n; i++) {
     sum +=i;
  }
 return sum;
}
```

n=3 (1+2+3)

0/p = 6

n = 5 (1+2+3+4+5)

o/p = 15

Analysis of Algorithms

- Order of growth

f(n) = 2n2+5n+1

g(n) = 6n+2

Direct ways.

1. Ignore lower order terms
2. Ignore leading constants

Which terms are lower order

Analysis of Algorithms

- Order of growth

f(n) = 2n2+5n+1

g(n) = 6n+2

Direct ways.

1. Ignore lower order terms
2. Ignore leading constants

Which terms are lower order

Which terms are lower order

C < log(logn) < log(n) < n1/3 <n1/2 <n < n2< n3 …< 2n< nn

Asymptotic notations

```
int calc(int[] arr) {
  int sum =0;
  for(int i:arr) {
    sum += arr[i];
  }
  return sum;
}
```

Asymptotic notations

```
int calc1(int[] arr) {
  if(arr.length %2 == 0)
    return 0;


  int sum =0;
  for(int i:arr) {
    sum += arr[i];
  }
  return sum;
}
```

Asymptotic notations

```
int calc(int[] arr) {
    int sum =0;
    for(int i:arr) {
        sum += arr[i];
    }
    return sum;
}
```

```
int calc1(int[] arr) {
    if(arr.length %2 == 0)
        return 0;

    int sum =0;
    for(int i:arr) {
        sum += arr[i];
    }
    return sum;
}
```

Best Case = Constant

Average case = Linear

Worst case = Linear

Asymptotic notations

- Big O : Exact or upper bond
- Theta : Exact bond
- Omega : Exact or lower bond

Asymptotic notations

- Big O : Exact or upper bond

  Represents Order of growth

  Example:

```
int linearSearch(int arr, int n, int k) {
  for(int i=0; i<n; i++) {
      if(arr[i] == k)
        return i;
       }
  return -1;
}
```

Asymptotic notations

Omega : Exact or lower bond

Example: quick sort.
nlogn

Asymptotic notations

Theta : Exact bond

Example: N player game.
Minimum time required to initialize the game.

Analysis of loops

```
//example1
void function1(int n) {
   for(int i=0; i<n; i++) {
      // constant work
   }
}
```

Analysis of loops

```
//example1
void function1(int n) {
   for(int i=0; i<n; i++) {
      // constant work
   }
}
```

```
//example2
void function2(int n) {
   for(int i=0; i<n; i*c) {
      // constant work
   }
}
```

Analysis of loops

```
//example1
void function1(int n) {
   for(int i=0; i<n; i++) {
      // constant work
   }
}
```

```
//example2
void function2(int n) {
   for(int i=0; i<n; i*2) {
      // constant work
   }
}
```

```
//example3
void function3(int n) {
   for(int i=2; i<n; i=pow(i,c)) {
      // constant work
   }
}
```

Analysis of loops

```
//example3
void function3(int n) {
  for(int i=2; i<n; i=pow(i,c)) {
     // constant work
  }
}
```

Analysis of loops

```
void function4(int n) {
for(int i=0; i<n; i++) {
// some constant work
}
for(int i=0; i<n; i++) {
// some constant work
}
for(int i=0; i<100; i++) {
//some constant work
}
}
```

Analysis of loops

```
void function5(int n) {
for(int i=0; i<n; i++) {
   for(int i=0; i<n; i++) {
   // some constant work
   }
 }
}
```

Mathematics

- Finding the number of digits in a number.

Mathematics

- Finding the number of digits in a number.

```
public static int findDigits(int n) {
   int sum =0;
   while(n != 0) {
      n = n/10;
      sum++;
   }
return sum;
}
```

Mathematics

- Finding the number of digits in a number.

```
public static int findDigitsRec(int n) {
   if(n == 0)
   return 0;

   return 1+ findDigitsRec(n/10);
}
```

Mathematics

- Factorial of a number

Mathematics

- Factorial of a number

```
public static int factorial(int n) {
   if(n==0) return 1;
   return n*factorial(n-1);
}
```

Mathematics

- Check if number is prime

Mathematics

- Check if number is prime

```
public static boolean checkPrime(int n) {
    boolean prime = true;
    for (int j = 2; j <= n / 2; j++)
        {
            if (n % j == 0)
            {
            prime = false;
            break;
            }
       }
    return prime;
}
```

Mathematics

- Prime Numbers.

Mathematics

- Prime Numbers.

```
static void printPrimeNumber(int n) {
   boolean prime = true;
    for (int i = 1; i <= n; i++) {
        if (i == 1 || i == 0)  continue;

        for (int j = 2; j <= i / 2; j++) {
            if (i % j == 0) {
               prime = false;
                break;
            }
        }
        if (prime)
          System.out.print(i + " ");
    }
}
```

Mathematics

- Ispalindrome

Mathematics

- Ispalindrome

```
public static boolean ispalindrome(int n) {
  int r = 0,reverse=0,temp;
  temp=n;
  while(n>0) {
        r = n%10;  //getting remainder
        reverse = (reverse*10)+r;
        n=n/10;
  }
  if(temp==reverse)
        return true;

  return false;
}
```

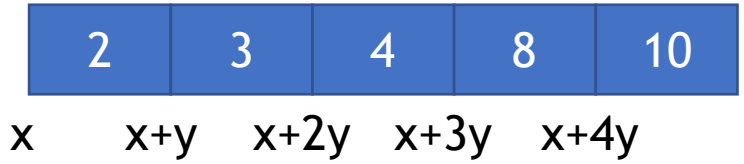
Arrays

```
int arr[] = {2, 3, 4, 8, 10};
int arr[] = new int[size];
```

Arrays

```
int arr[] = {2, 3, 4, 8, 10};
```

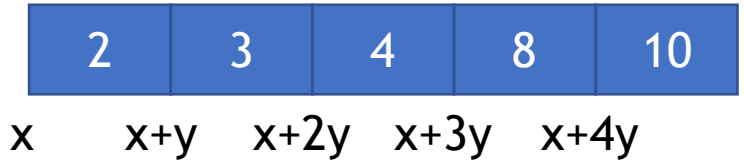
Contiguous memory



Arrays

```
int arr[] = {2, 3, 4, 8, 10};
```

Contiguous memory



Advantages
1. Random Access
2. Cache friendliness

Arrays

Different types

1. **Fixed size**

```
int arr[] = new int[10];
```

2. **Dynamic size arrays**

```
ArrayList list = new ArrayList();
```

Arrays

Operations on Array

1. Searching an item

```
public static int serach(int arr[], int n, int item) {
   for(int i=0; i<n ; i++) {
      if(arr[i] == item)
            return i;
      }
   return -1;
}
```

Arrays

Operations on Array

1. Insertion

Arrays

Operations on Array

1. Insertion

```
public static int insert(int array[], int n, int cap, int
index, int item) {
   if(n == cap) return n;
   for(int i=n-1; i < index; i--) {
      array[i + 1] = array[i];
   }

   array[index] = item;
   return n+1;
}
```

Arrays

Operations on Array

1. **Insertion – O(n)**

2. **Deletion – O(n)**

3. **Search – O(n) //Unsorted**
   Search – O(logn) //sorted

get(i) – O(1)
Set(i) //update(i) = O(1)

Arrays

1. Check if array is sorted

Arrays

1. **Check if array is sorted**

```
public static boolean isSorted(int array[], int n) {
   for(int i=0; i<n; i++) {
      for (int j=i+1; j<n; j++) {
       if(array[j] < array[i])
          return false;
       }
      }
   return true;
}
```

Arrays

1. Check if array is sorted

a0, a1, a2 … ai-1, ai, … an

ai-1 < ai

Arrays

1. Check if array is sorted

a0, a1, a2 … ai-1, ai, … an

```
public static boolean isSortedEff(int array[], int n) {
   for(int i=1; i<n; i++) {
    if(array[i] < array[i-1])
      return false;
    }
   return true;
}
```

Arrays

2. Reverse an array

Arrays

2. Reverse an array

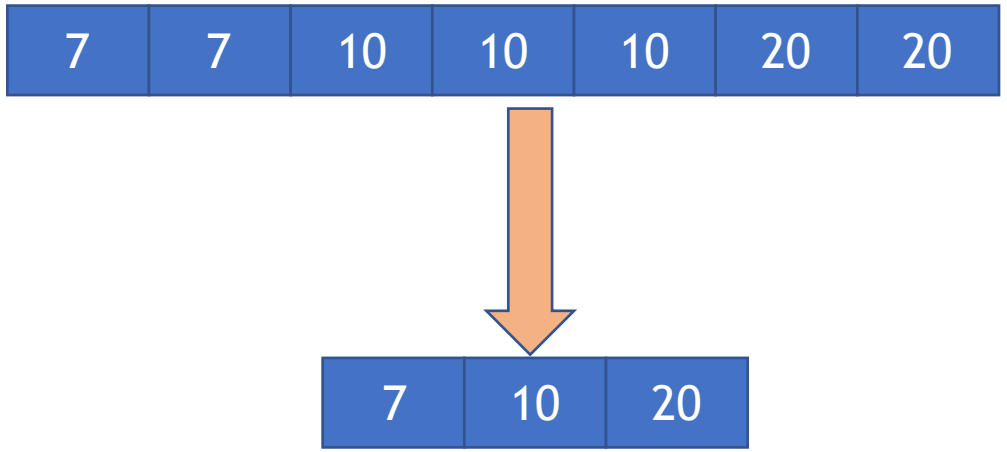
```
public static void reverse(int arr[], int n) {
   int low = 0, high = n-1;
   while (low < high) {
      int temp = arr[low];
      arr[low] = arr[high];
      arr[high] = temp;

      low++;
      high--;
   }
}
```

Arrays

3. Remove Duplicate from sorted array

Arrays

3. Remove Duplicate from sorted array



Arrays

4. Left Rotate an array by 1 space

[2,5,7,8,9]

[5,7,8,9,2]

Arrays

4. Left Rotate an array by 1 space

```
public static void leftRotate(int arr[], int n) {
   int temp = arr[0];

   for(int i=1; i<n; i++) {
      arr[i-1] = arr[i];
   }

   arr[n-1] = temp;
}
```

Arrays

4. Left Rotate an array by 1 space

```
public static void leftRotate(int arr[], int n) {
   int temp = arr[0];

   for(int i=1; i<n; i++) {
      arr[i-1] = arr[i];
   }

   arr[n-1] = temp;
}
```

O(n)

Arrays

5. Left Rotate an array by m spaces

m=2

[2,5,7,8,9]

[7,8,9,2,5]

Arrays

5. Left Rotate an array by m space

```
public static void rotate(int arr[], int n, int m) {
   for(int i=0; i<m; i++) {
      leftRotate(arr, n);
   }
}
```

O(mn)

Arrays

5. Left Rotate an array by m space

Better solution,

```
public static void rotateBetter(int arr[], int n, int m) {
    reverseArray(arr, 0, m-1);
    reverseArray(arr, m, n-1);
    reverseArray(arr, 0, n-1);
}


public static void reverseArray(int arr[], int low, int high) {
    while(low < high) {
        int temp = arr[low];
        arr[low] = arr[high];
        arr[high] = temp;

        low++;
        high--;
    }
}
```

Arrays

5. Left Rotate an array by m space

Better solution,

```
public static void rotateBetter(int arr[], int n, int m) {
    reverseArray(arr, 0, m-1);
    reverseArray(arr, m, n-1);
    reverseArray(arr, 0, n-1);
}


public static void reverseArray(int arr[], int low, int high) {
    while(low < high) {
        int temp = arr[low];
        arr[low] = arr[high];
        arr[high] = temp;

        low++;
        high--;
    }
}
```

O (2n) -> **O**(n)

Arrays

6. Leaders in an array

[10,7,8,9,5,3]

[10,9,5,3]

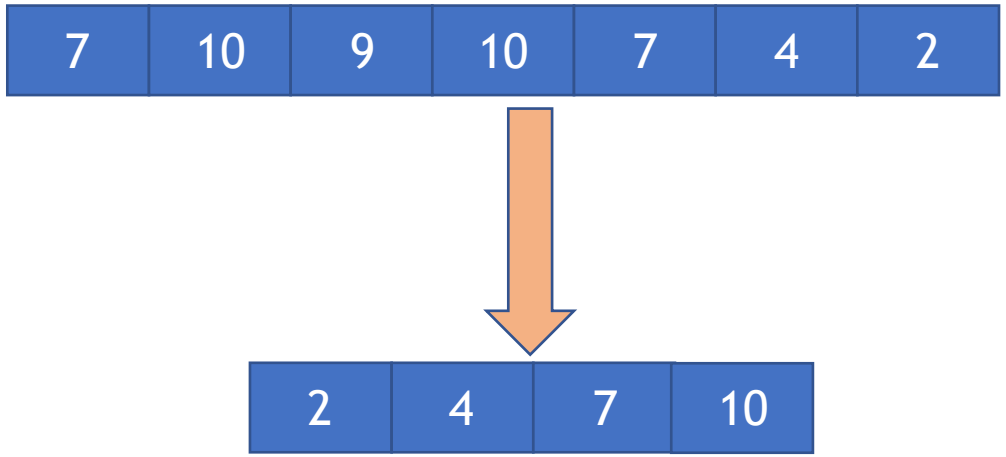
Arrays

6. Leaders in an array

```
public static void leaders(int arr[], int n) {
   for(int i=0; i<n; i++) {
   boolean flag = false;

   for(int j=i+1; j<n; j++) {
      if(arr[i] <= arr[j]) {
         flag = true;
         break;
      }
   }
   if(!flag)
      System.out.println(arr[i]);
   }
}
```

Arrays

6. Leaders in an array / in theta(n)



Arrays

7. Stock buy and sell problem

Array = {1,4,3,8,11}

Arrays

8. Sliding window technique

Array = {1,7,20,-8,13}
K = 3
o/p – 28

Array = {4, -12, 6, 70, 18}
K = 2
o/p - 88

References

- Mathematics -  FindingDigits.java
- Arrays -  ArrayOperations.java