

# ALGORITHMS & COMPLEXITY

CS203.3

Dr. Rasika Ranaweera

## Lab Exercise I 2021 Summer

**Name:** L D T Senevirathne

**Student No:** 19170

## Lab Exercise (Algorithm Revision)

### Important:

- ✓ Objective is to understand, implement, and analyze familiar algorithms
- ✓ Bring your answer sheets (on top write your name, ID, course code) to the next class
- ✓ Learn from others or Google but DO NOT COPY

### Introduction

Searching algorithms aim to find position of a target value within an array/list. Selection, merge, linear, binary, jump, ternary search are examples. Lets discuss about two of them.

#### Linear Search

Checks each element of the list until a match is found or reaches end

#### Binary search

Compares the target value to the middle element within a sorted array and eliminates irrelevant half

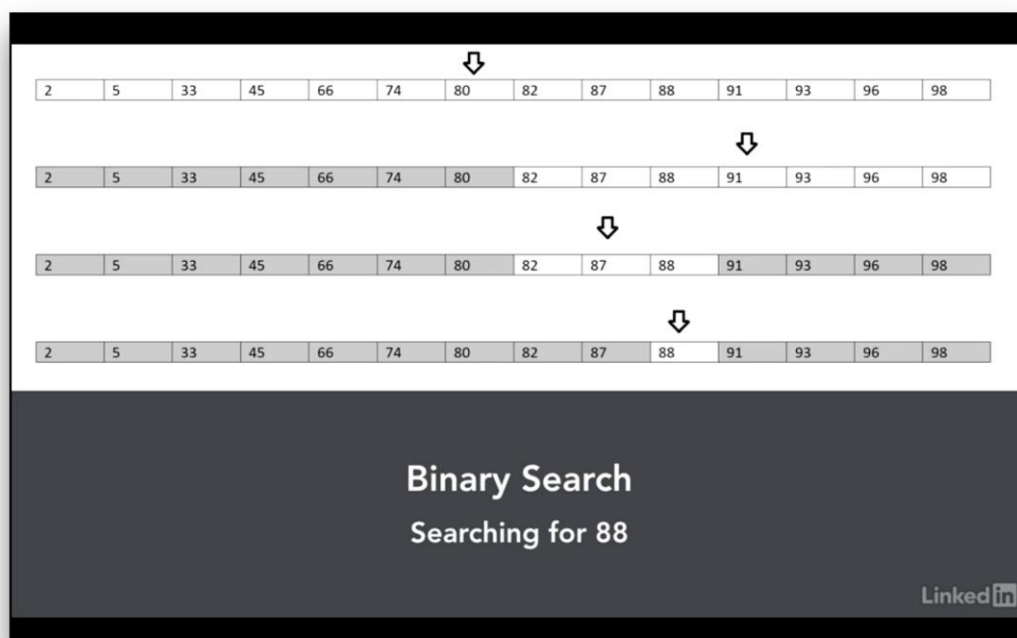


Image courtesy: <https://www.lynda.com/C-tutorials/Binary-search-explained/604241/636317-4.html>

## Assignment:

Prepare a report answering the basic questions [R], then write programs [S] using any programming language, and finally plot the data on an excel file [E].

1. Write pseudo-codes for the above searching algorithms (you must also include references if used) [S].

### Linear Search Pseudocode

```
function linearSearch (items, target)
    for i from 0 to (length-1)
        if items[i] == target
            then return i
        end if
    end for
end
```

### Binary Search Pseudocode

```
function binarySearch (items, target, left, right)
    left = 0;
    right = items.length -1;
    mid = (left + right) / 2

    while left<=right
        if items[mid] == target
            then return mid;
        else if item[mid] < target
            then right = mid -1;
        else item[mid] > target
            then left = mid + 1;
        end if
    end while
end
```

2. Implement both algorithms using any program language [S]
  - a. Write a program to generate any number of random integers in 0 to 100 range. Your program should get the size as a parameter and return the numbers as an array [S].

```
package com.linearsearch;

import java.util.*;

public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("Array size:\t");
        int n = sc.nextInt();

        ArrayList list = GenerateRandomIntegers (n, 0, n);

        //printing the array
        StringBuffer sb = new StringBuffer();
        for (Object s : list) {
            sb.append(s);
            sb.append(" ");
        }
        String str = sb.toString();
        System.out.println(str);
    }

    public static ArrayList GenerateRandomIntegers(int size, int min, int
max) {
        ArrayList numbers = new ArrayList();
        Random rand = new Random();

        while (numbers.size() < size) {
            //getting random numbers within range
            int randomNumber = rand.nextInt((max - min) + 1) + min;
            //check for duplicates
            if (!numbers.contains(randomNumber)) {
                numbers.add(randomNumber);
            }
        }
        return numbers;
    }
}
```

- b. Now implement the linear search. Pass the random array of 100 items as the list and "50" as the item to find [S].

```
package com.linearsearch;

import java.util.*;

public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("Array size:\t");
        int n = sc.nextInt();
        System.out.println("Element to be found: ");
        int x = sc.nextInt();

        ArrayList list = GenerateRandomIntegers (n, 0, n);

        //printing the array
        StringBuffer sb = new StringBuffer();
        for (Object s : list) {
            sb.append(s);
            sb.append(" ");
        }
        String str = sb.toString();
        System.out.println(str);

        linearSearch(list, x);
    }

    public static ArrayList GenerateRandomIntegers(int size, int min, int
max) {
        ArrayList numbers = new ArrayList();
        Random rand = new Random();

        while (numbers.size() < size) {
            //getting random numbers within range
            int randomNumber = rand.nextInt((max - min) + 1) + min;
            //check for duplicates
            if (!numbers.contains(randomNumber)) {
                numbers.add(randomNumber);
            }
        }
        return numbers;
    }

    public static void linearSearch(ArrayList arr, int target) {
        for(int i=0;i<arr.size();i++) {
            if(arr.get(i).equals(target)) {
                System.out.println("Item found at = "+(i+1));
            }
        }
    }
}
```

- c. Compute the time to find 50 in the array and record the time consumed [S].

```
package com.linearsearch;

import java.util.*;

public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("Array size:\t");
        int n = sc.nextInt();
        System.out.println("Element to be found: ");
        int x = sc.nextInt();

        ArrayList list = GenerateRandomIntegers (n, 0, n);

        //printing the array
        StringBuffer sb = new StringBuffer();
        for (Object s : list) {
            sb.append(s);
            sb.append(" ");
        }
        String str = sb.toString();
        System.out.println(str);

        //start computing time
        long startTime = System.nanoTime();

        linearSearch(list, x);

        //end computing time
        long endTime = System.nanoTime();
        long timeElapsed = endTime - startTime;
        System.out.println("Execution time in nanoseconds: " + timeElapsed);
        System.out.println("Execution time in milliseconds: " + timeElapsed
/ 1000000);
    }

    public static ArrayList GenerateRandomIntegers(int size, int min, int
max) {
        ArrayList numbers = new ArrayList();
        Random rand = new Random();

        while (numbers.size() < size) {
            //getting random numbers within range
            int randomNumber = rand.nextInt((max - min) + 1) + min;
            //check for duplicates
            if (!numbers.contains(randomNumber)) {
                numbers.add(randomNumber);
            }
        }
        return numbers;
    }

    public static void linearSearch(ArrayList arr, int target) {
        for(int i=0;i<arr.size();i++) {
            if(arr.get(i).equals(target)) {
                System.out.println("Item found at = "+(i+1));
            }
        }
    }
}
```

```
}  
}  
}
```

```
Run: Main x  
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2020.3.  
Array size:  
100  
Element to be found:  
50  
96 85 26 7 58 86 81 91 99 9 12 66 22 50 38 6 47 98 33 73 14 39 36 17 43 21 87 57 63 76 59 1 82 29 51 93 37 6  
Item found at = 14  
Execution time in nanoseconds: 7679000  
Execution time in milliseconds: 7.679  
  
Process finished with exit code 0  
|  
  
Run Debug TODO Problems Profiler Terminal Build
```

d. Conduct this test for 10 times and plot the timing on a graph [E].

```
package com.linearsearch;

import java.util.*;

public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("Array size:\t");
        int n = sc.nextInt();
        System.out.println("Element to be found: ");
        int x = sc.nextInt();

        //looping for 10 times
        for(int i = 0; i < 10; i++) {
            System.out.println("\nArray: " + (i+1));
            ArrayList list = GenerateRandomIntegers(n, 0, n);

            //printing the array
            StringBuffer sb = new StringBuffer();
            for (Object s : list) {
                sb.append(s);
                sb.append(" ");
            }
            String str = sb.toString();
            System.out.println(str);

            //start computing time
            long startTime = System.nanoTime();

            linearSearch(list, x);

            //end computing time
            long endTime = System.nanoTime();
            long timeElapsed = endTime - startTime;
            System.out.println("Execution time in nanoseconds: " +
timeElapsed);
            System.out.println("Execution time in milliseconds: " +
timeElapsed / 1000000.0);
        }
    }

    public static ArrayList GenerateRandomIntegers(int size, int min, int
max) {
        ArrayList numbers = new ArrayList();
        Random rand = new Random();

        while (numbers.size() < size) {
            //getting random numbers within range
            int randomNumber = rand.nextInt((max - min) + 1) + min;
            //check for duplicates
            if (!numbers.contains(randomNumber)) {
                numbers.add(randomNumber);
            }
        }
        return numbers;
    }

    public static void linearSearch(ArrayList arr, int target) {
        for(int i=0;i<arr.size();i++) {
            if(arr.get(i).equals(target)) {
                System.out.println("Item found at = " + (i+1));
            }
        }
    }
}
```



```
}  
}  
}  
}
```

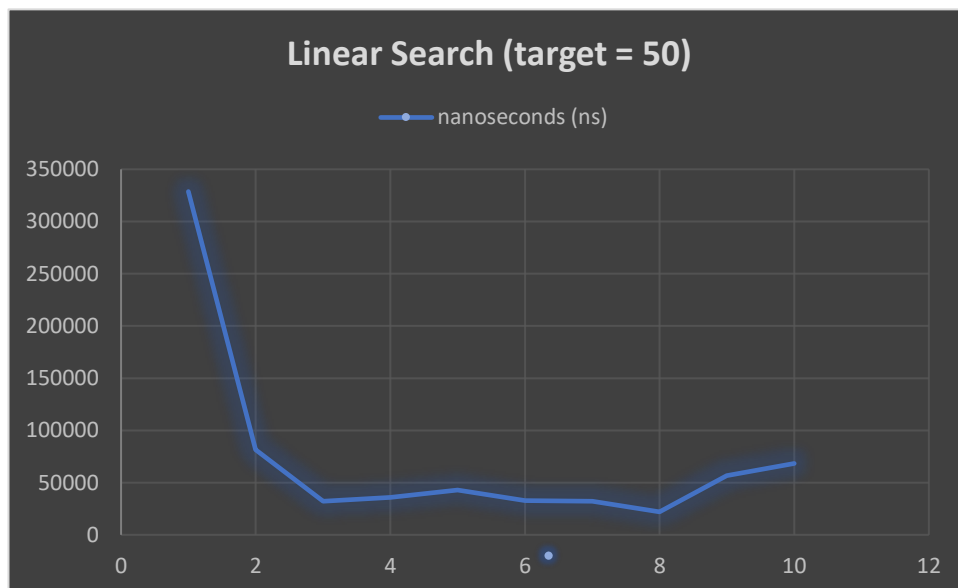
**Array size = 100**

Linear Search (Array size = 100)

No	nanoseconds (ns)	milliseconds (ms)	Item found at
1	328700	0.3287	74
2	81600	0.0816	94
3	32100	0.0321	56
4	35700	0.0357	100
5	42900	0.0429	34
6	32700	0.0327	29
7	32100	0.0321	82
8	22000	0.022	86
9	56700	0.0567	82
10	68300	0.0683	67

average = 73280

Average = 73280 ns



- e. Now conduct the same test for 1K (1000), 5K, 10K, ... , 50K and plot the average timing [E].

**Array size = 1000**

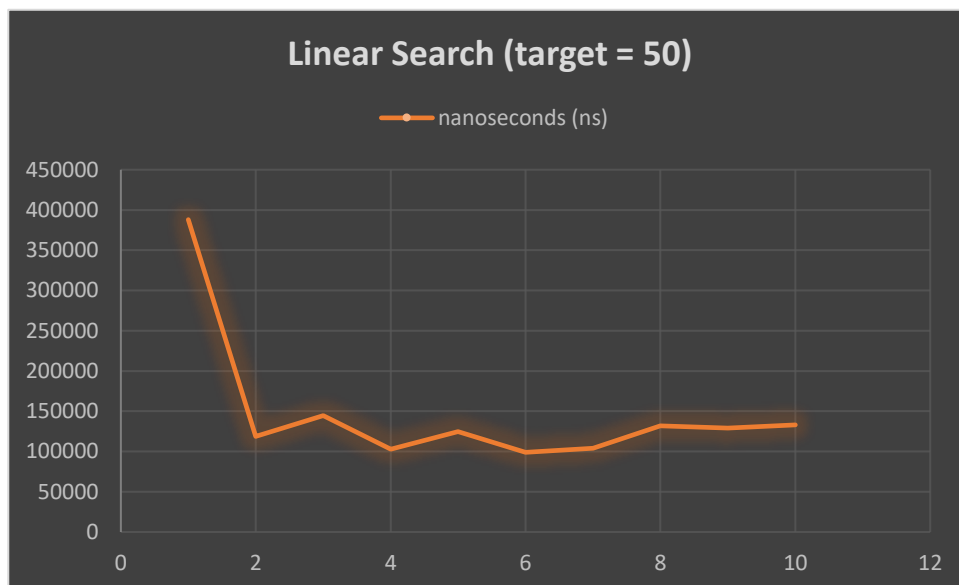
Linear Search (Array size = 1000)

No	nanoseconds (ns)	milliseconds (ms)	Item found at
1	388000	0.388	555
2	118700	0.1187	378
3	144800	0.1448	449
4	103000	0.103	233
5	124900	0.1249	910
6	99000	0.099	52
7	104100	0.1041	307
8	132000	0.132	366
9	129300	0.1293	180
10	133100	0.1331	611

average

= 147690

Average = 147690 ns



**Array size = 5000**

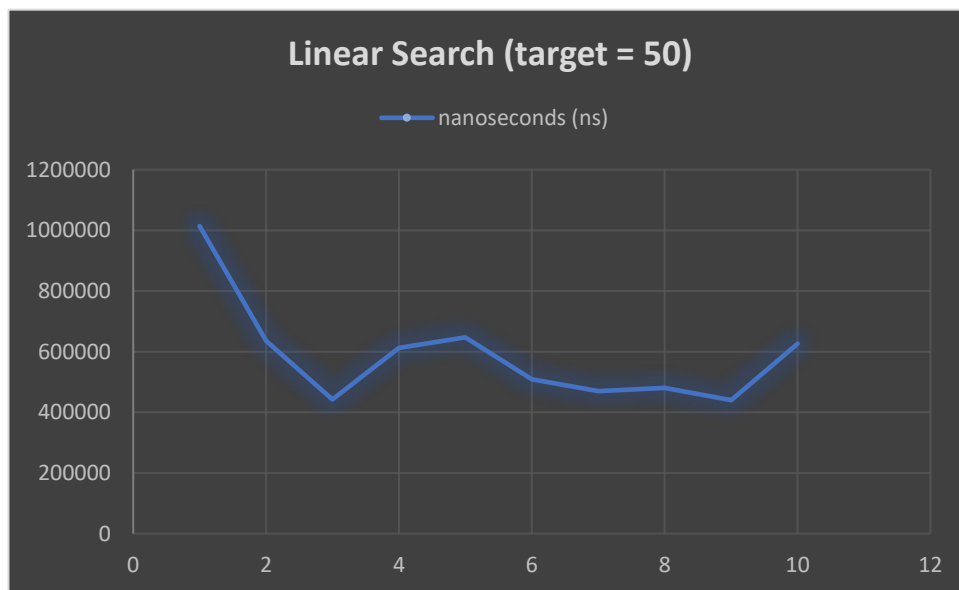
Linear Search (Array size = 5000)

No	nanoseconds (ns)	milliseconds (ms)	Item found at
1	1013700	1.0137	2723
2	635800	0.6358	2175
3	442500	0.4425	3216
4	612000	0.612	4006
5	647300	0.6473	640
6	508600	0.5086	164
7	469600	0.4696	3046
8	479900	0.4799	3500
9	439900	0.4399	3539
10	625500	0.6255	3312

average

= 587480

Average = 587480 ns



**Array size = 10000**

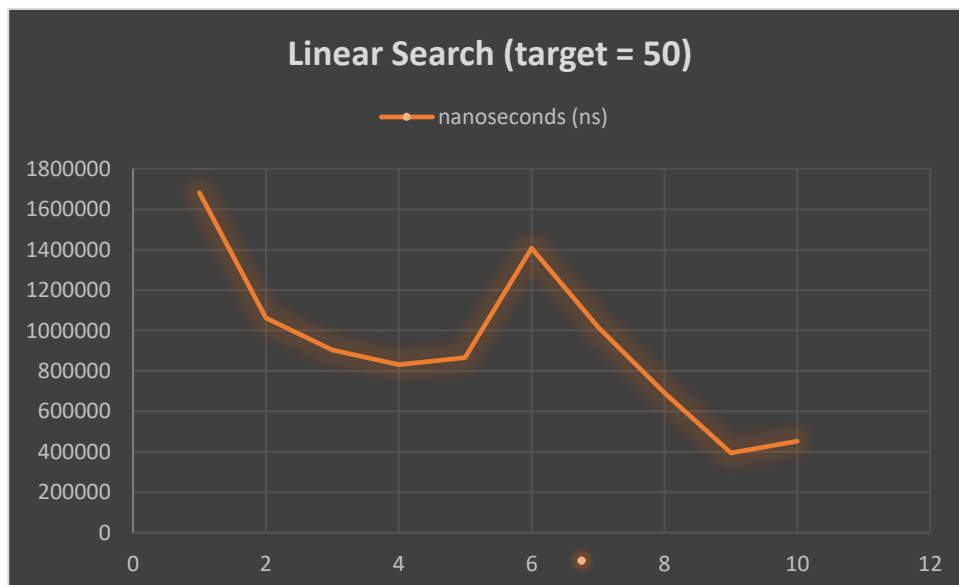
Linear Search (Array size = 10000)

No	nanoseconds (ns)	milliseconds (ms)	Item found at
1	1681700	1.6817	6037
2	1062200	1.0622	865
3	903700	0.9037	4433
4	831400	0.8314	2584
5	865600	0.8656	1491
6	1407300	1.4073	7188
7	1016900	1.0169	1601
8	692100	0.6921	2104
9	394900	0.3949	5129
10	453300	0.4533	630

average

= 930910

Average = 930910 ns



**Array size = 50000**

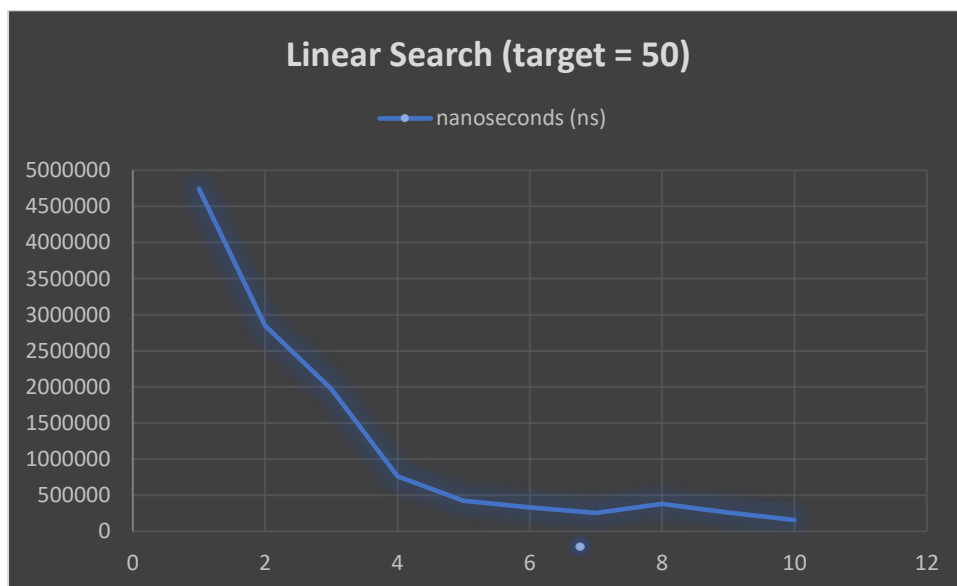
Linear Search (Array size = 50000)

No	nanoseconds (ns)	milliseconds (ms)	Item found at
1	4742900	4.7429	45062
2	2849600	2.8496	8227
3	1968900	1.9689	33734
4	763300	0.7633	46800
5	425200	0.4252	24108
6	330900	0.3309	45418
7	255300	0.2553	29026
8	380700	0.3807	1378
9	261100	0.2611	30644
10	157800	0.1578	24187

average

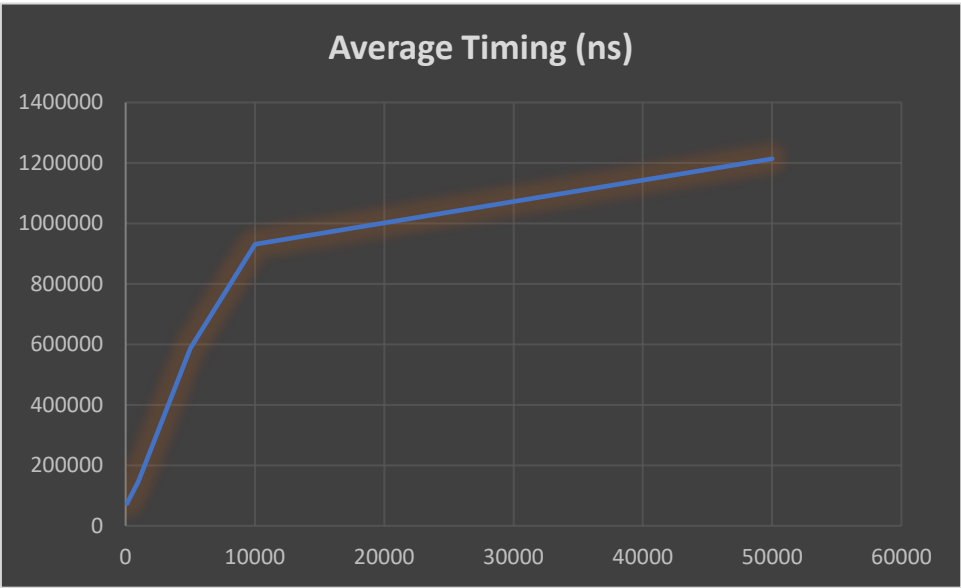
= 1213570

Average = 1213570 ns



**Average timing for Linear Search 100, 1k, 5k, 10k, 50k**

Array Size	Average Timing (ns)
100	73280
1000	147690
5000	587480
10000	930910
50000	1213570



3. Redo the same test for binary search.
  - a. Write a program to generate any number of random integers in 0 to 100 range. Your program should get the size as a parameter and return the numbers as an array [S].

```
package com. binarysearch;

import java.util.*;

public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("Array size:\t");
        int n = sc.nextInt();

        ArrayList list = GenerateRandomIntegers (n, 0, n);

        //printing the array
        StringBuffer sb = new StringBuffer();
        for (Object s : list) {
            sb.append(s);
            sb.append(" ");
        }
        String str = sb.toString();
        System.out.println(str);
    }

    public static ArrayList GenerateRandomIntegers(int size, int min, int
max) {
        ArrayList numbers = new ArrayList();
        Random rand = new Random();

        while (numbers.size() < size) {
            //getting random numbers within range
            int randomNumber = rand.nextInt((max - min) + 1) + min;
            //check for duplicates
            if (!numbers.contains(randomNumber)) {
                numbers.add(randomNumber);
            }
        }
        return numbers;
    }
}
```

- b. Now implement the binary search. Pass the random array of 100 items as the list and "50" as the item to find [S].

```
package com.binarysearch;

import java.util.*;

public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("Array size:\t");
        int n = sc.nextInt();
        System.out.println("Element to be found: ");
        int x = sc.nextInt();

        ArrayList list = GenerateRandomIntegers (n, 0, n);

        //printing the unsorted array
        System.out.println("Unsorted Array: ");
        StringBuffer sb = new StringBuffer();
        for (Object s : list) {
            sb.append(s);
            sb.append(" ");
        }
        String unsorted = sb.toString();
        System.out.println(unsorted);

        //sorting the ArrayList
        Collections.sort(list);

        //printing the sorted array
        System.out.println("Sorted Array: ");
        StringBuffer sb2 = new StringBuffer();
        for (Object s : list) {
            sb2.append(s);
            sb2.append(" ");
        }
        String sorted = sb2.toString();
        System.out.println(sorted);

        binarySearch(list, x);
    }

    public static ArrayList GenerateRandomIntegers(int size, int min,
int max) {
        ArrayList numbers = new ArrayList();
        Random rand = new Random();

        while (numbers.size()<size+1) {
            //getting random numbers within range
            int randomNumber = rand.nextInt((max-min)+1) + min;
            //check for duplicates
            if(!numbers.contains(randomNumber)) {
                numbers.add(randomNumber);
            }
        }
        return numbers;
    }

    public static int binarySearch(ArrayList<Integer> arr, int target) {
        int left = 0;
```



```
int right = arr.size() - 1;

while (left <= right) {
    int mid = (left + right)/2;
    if (arr.get(mid) == target) {
        System.out.println("Item found at: "+(mid+1));
        return mid+1;
    }
    else if (arr.get(mid) < target) {
        left = mid + 1;
    }
    else {
        right = mid - 1;
    }
}
return -1;
}
```

- c. Compute the time to find 50 in the array and record the time consumed [S].

```
package com.binarysearch;

import java.util.*;

public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("Array size:\t");
        int n = sc.nextInt();
        System.out.println("Element to be found: ");
        int x = sc.nextInt();

        ArrayList list = GenerateRandomIntegers (n, 0, n);

        //printing the unsorted array
        System.out.println("Unsorted Array: ");
        StringBuffer sb = new StringBuffer();
        for (Object s : list) {
            sb.append(s);
            sb.append(" ");
        }
        String unsorted = sb.toString();
        System.out.println(unsorted);

        //sorting the ArrayList
        Collections.sort(list);

        //printing the sorted array
        System.out.println("Sorted Array: ");
        StringBuffer sb2 = new StringBuffer();
        for (Object s : list) {
            sb2.append(s);
            sb2.append(" ");
        }
        String sorted = sb2.toString();
        System.out.println(sorted);

        //start computing time
        long startTime = System.nanoTime();

        binarySearch(list, x);

        //end computing time
        long endTime = System.nanoTime();
        long timeElapsed = endTime - startTime;
        System.out.println("Execution time in nanoseconds: " + timeElapsed);
        System.out.println("Execution time in milliseconds: " + timeElapsed
/ 1000000);

    }

    public static ArrayList GenerateRandomIntegers(int size, int min, int
max) {
        ArrayList numbers = new ArrayList();
        Random rand = new Random();

        while (numbers.size() < size + 1) {
            //getting random numbers within range
            int randomNumber = rand.nextInt((max - min) + 1) + min;
            //check for duplicates
        }
    }
}
```

```

        if(!numbers.contains(randomNumber)) {
            numbers.add(randomNumber);
        }
    }
    return numbers;
}

public static int binarySearch(ArrayList<Integer> arr, int target) {
    int left = 0;
    int right = arr.size() - 1;

    while (left <= right) {
        int mid = (left + right)/2;
        if (arr.get(mid) == target) {
            System.out.println("Item found at: " + (mid+1));
            return mid+1;
        }
        else if (arr.get(mid) < target) {
            left = mid + 1;
        }
        else {
            right = mid - 1;
        }
    }
    return -1;
}
}

```

```

"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2020.3.1\lib\idea_rt.jar=50432:C:\Program Files\JetBrains\IntelliJ IDEA 2020.3.1\bin" -Dfile.encoding=UTF-8
Array size:
100
Element to be found:
50
Unsorted Array:
42 70 83 51 58 15 73 57 37 87 29 14 89 39 100 22 30 45 54 97 98 47 28 62 82 7 59 75 53 95 24 81 20 8 41 48 17 74 34 86 93 68 33 0 56 61 1 49 99 76 12 69 65 6 66 18 64
Sorted Array:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58
Item found at: 51
Execution time in nanoseconds: 5581500
Execution time in milliseconds: 5.5815

Process finished with exit code 0
|

```

d. Conduct this test for 10 times and plot the timing on a graph [E].

```
package com.binarysearch;

import java.util.*;

public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("Array size:\t");
        int n = sc.nextInt();
        System.out.println("Element to be found: ");
        int x = sc.nextInt();

        //looping for 10 times
        for(int i = 0; i < 10; i++) {
            System.out.println("\nArray: " + (i + 1));

            ArrayList list = GenerateRandomIntegers(n, 0, n);

            //printing the unsorted array
            System.out.println("Unsorted Array: ");
            StringBuffer sb = new StringBuffer();
            for (Object s : list) {
                sb.append(s);
                sb.append(" ");
            }
            String unsorted = sb.toString();
            System.out.println(unsorted);

            //sorting the ArrayList
            Collections.sort(list);

            //printing the sorted array
            System.out.println("Sorted Array: ");
            StringBuffer sb2 = new StringBuffer();
            for (Object s : list) {
                sb2.append(s);
                sb2.append(" ");
            }
            String sorted = sb2.toString();
            System.out.println(sorted);

            //start computing time
            long startTime = System.nanoTime();

            binarySearch(list, x);

            //end computing time
            long endTime = System.nanoTime();
            long timeElapsed = endTime - startTime;
            System.out.println("Execution time in nanoseconds: " +
timeElapsed);
            System.out.println("Execution time in milliseconds: " +
timeElapsed / 1000000.0);
        }
    }

    public static ArrayList GenerateRandomIntegers(int size, int min, int
max) {
        ArrayList numbers = new ArrayList();
        Random rand = new Random();

        while (numbers.size() < size + 1) {
```

```

        //getting random numbers within range
        int randomNumber = rand.nextInt((max-min)+1) + min;
        //check for duplicates
        if(!numbers.contains(randomNumber)) {
            numbers.add(randomNumber);
        }
    }
    return numbers;
}

public static int binarySearch(ArrayList<Integer> arr, int target) {
    int left = 0;
    int right = arr.size() - 1;

    while (left <= right) {
        int mid = (left + right)/2;
        if (arr.get(mid) == target) {
            System.out.println("Item found at: "+(mid+1));
            return mid+1;
        }
        else if (arr.get(mid) < target) {
            left = mid + 1;
        }
        else {
            right = mid - 1;
        }
    }
    return -1;
}
}

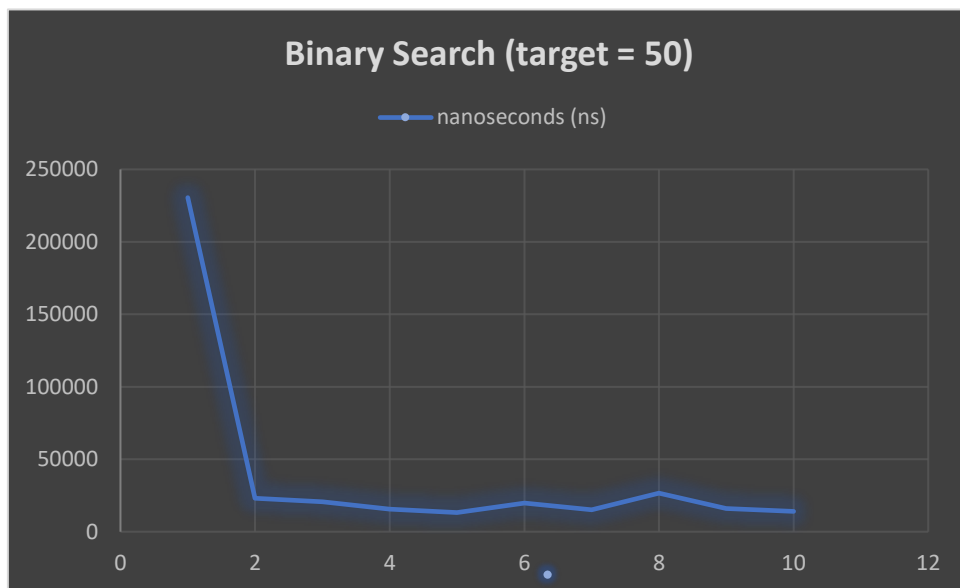
```

Binary Search (Array size = 100)

No	nanoseconds (ns)	milliseconds (ms)
1	230500	0.2305
2	23100	0.0231
3	20600	0.0206
4	15600	0.0156
5	13200	0.0132
6	19800	0.0198
7	15200	0.0152
8	26500	0.0265
9	16000	0.016
10	14100	0.0141

average = 39460

Average = 39460 ns



- e. Now conduct the same test for 1K (1000), 5K, 10K, ... , 50K and plot the average timing [E].

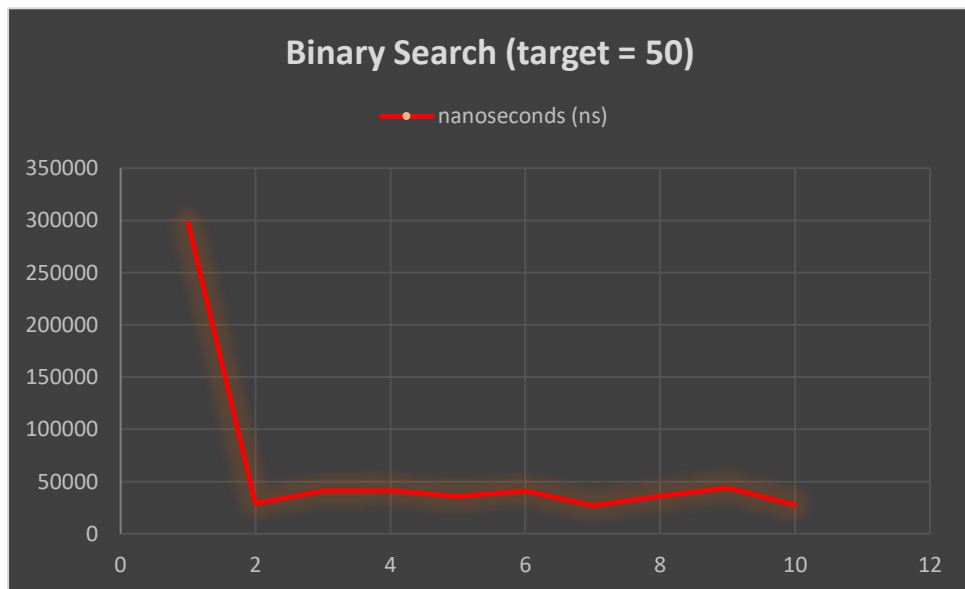
Binary Search (Array size = 1000)

No	nanoseconds (ns)	milliseconds (ms)
1	297500	0.2975
2	28900	0.0289
3	40900	0.0409
4	41400	0.0414
5	35800	0.0358
6	41000	0.041
7	27000	0.027
8	36000	0.036
9	44200	0.0442
10	27400	0.0274

average

= 62010

Average = 62010 ns

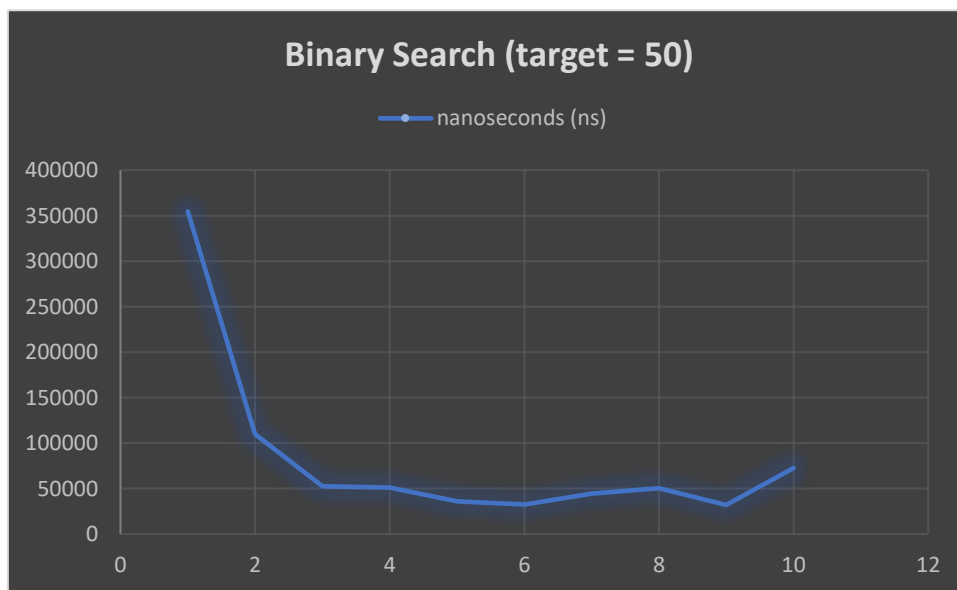


Binary Search (Array size = 5000)

No	nanoseconds (ns)	milliseconds (ms)
1	354800	0.3548
2	109800	0.1098
3	52700	0.0527
4	51100	0.0511
5	36100	0.0361
6	32700	0.0327
7	44400	0.0444
8	50500	0.0505
9	32000	0.032
10	72600	0.0726

average = 83670

Average = 83670 ns





Binary Search (Array size = 10000)

No	nanoseconds (ns)	milliseconds (ms)
1	363100	0.3631
2	61800	0.0618
3	77100	0.0771
4	33300	0.0333
5	75700	0.0757
6	50000	0.05
7	97000	0.097
8	43200	0.0432
9	46000	0.046
10	52800	0.0528

average

= 90000

Average = 90000 ns

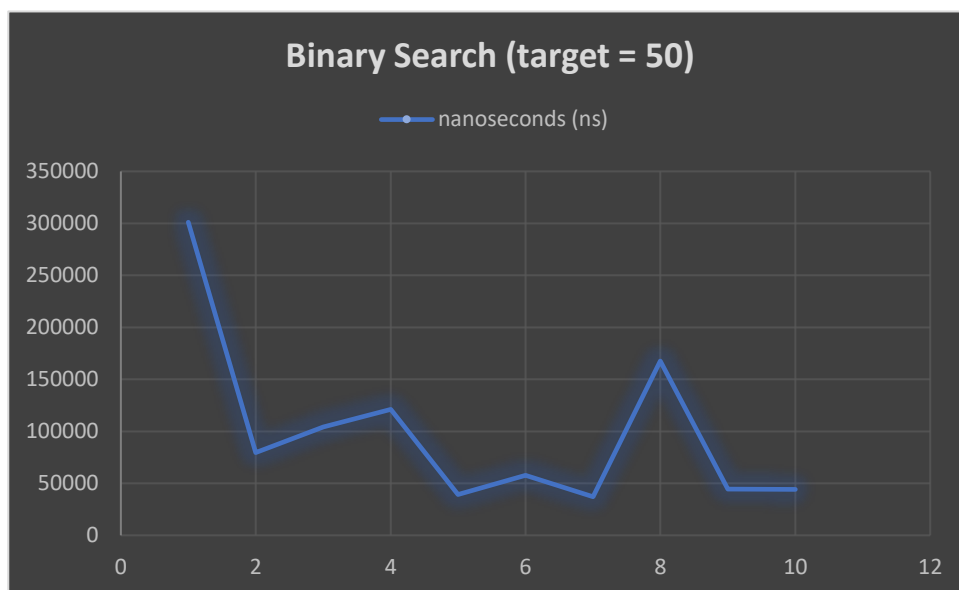


Binary Search (Array size = 50000)

No	nanoseconds (ns)	milliseconds (ms)
1	301100	0.3011
2	79600	0.0796
3	104200	0.1042
4	121000	0.121
5	39100	0.0391
6	57700	0.0577
7	37100	0.0371
8	167700	0.1677
9	44600	0.0446
10	44300	0.0443

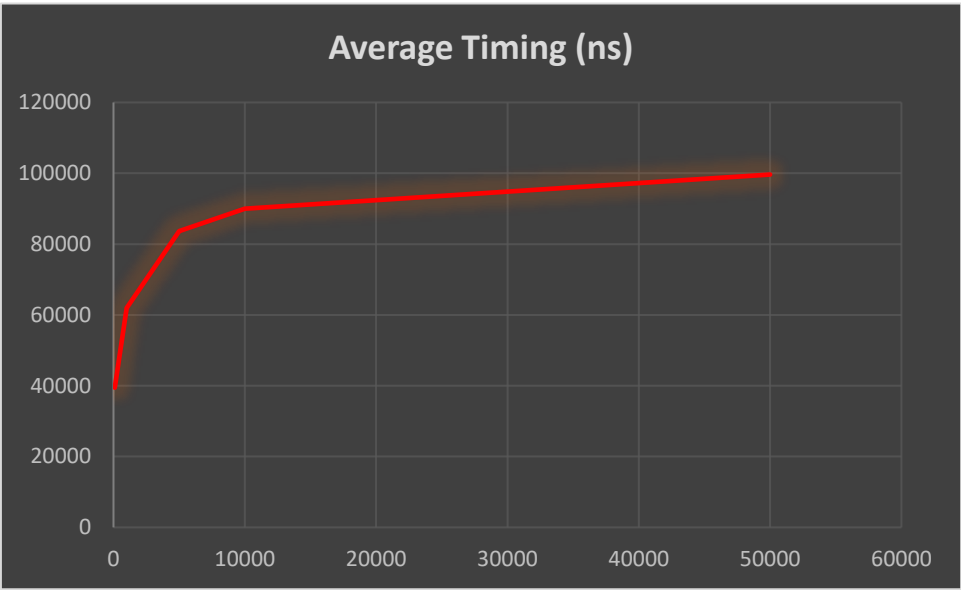
average = 99640

Average = 99640 ns

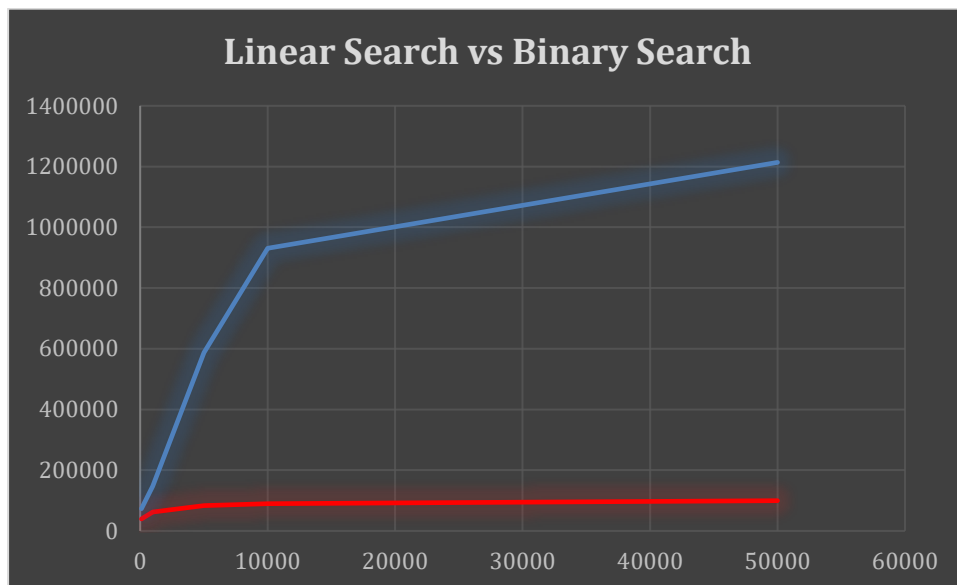


**Average timing for Binary Search 100, 1k, 5k, 10k, 50k**

Array Size	Average Timing (ns)
100	39460
1000	62010
5000	83670
10000	90000
50000	99640



4. Plot your finding on a graph (Red: Binary, Blue: Linear) and a table (including your computer specs) [E]



#### Computer Specifications

Device name	LAPTOP-CG9042HO
Processor	Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz 1.50 GHz
Installed RAM	8.00 GB (7.78 GB usable)
Device ID	0CE01049-9BB9-4A1E-A6A6-8D450C0C6BD6
Product ID	00327-35863-49960-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display
Edition	Windows 10 Home Single Language
Version	21H1
OS build	19043.1165

**References:**

[http://python-textbok.readthedocs.io/en/1.0/Sorting\\_and\\_Searching\\_Algorithms.html](http://python-textbok.readthedocs.io/en/1.0/Sorting_and_Searching_Algorithms.html)

# ALGORITHMS & COMPLEXITY

CS203.3

Dr. Rasika Ranaweera

## Assignment II

**Type:** Individual & Mandatory

**Duration:** 2 Hours (+ Homework)

## Lab Exercise (Algorithm Revision)

### Important:

- ✓ Objective is to understand, implement, and analyze familiar algorithms.
- ✓ Hand over your answer sheets (on top write your name, ID, course code) to an instructor.
- ✓ Save your programs under **home/cs203.3/** directory in your computer.
- ✓ Learn from others, read books, or Google but DO NOT COPY 1-to-1.

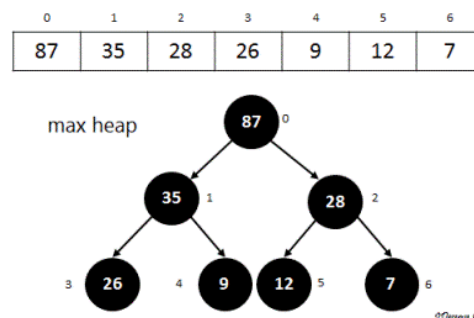
### Introduction:

Sorting is about ordering objects. We can distinguish different types of sorting:

- Internal sorting: A small no of objects that can fit into the main memory.
- External sorting: A large no of objects that require external storage during the sort.
- Stable, oblivious, sort by address etc.

### Assignment:

1. Write algorithms, pseudo-codes, and source codes (using any language you are comfortable) for following sorting algorithms and improve your programs.
  - i. Use illustrations to explain **selection** sort and **bucket** sort algorithms (like the following for heap sort algorithm).



(source: <https://www.ideserve.co.in/learn/heap-sort>)

## Selection Sort

01

i) Selection Sort

0	1	2	3	4	5	6
87	35	28	26	9	12	7

~~$i=0$  87 35~~

$\text{min} = 0$

$\text{min} = 87$  35 28 26 9 12 7

$i=0$  87 35 28 26 9 12 7  
 $i=0$

$i=1$  7 35 28 26 9 12 87  
 $i=1$

$i=2$  7 9 28 26 35 12 87  
 $i=2$

$i=3$  7 9 12 26 35 28 87  
 $i=3$

$i=4$  7 9 12 26 35 28 87  
 $i=4$

$i=5$  7 9 12 26 28 35 87  
 $i=5$

$i=6$  7 9 12 26 28 35 87  
 $i=6$

7	9	12	26	28	35	87
---	---	----	----	----	----	----



## Bucket Sort

0	1	2	3	4	5	6
87	35	28	26	9	12	7

Max = 87

Array size = Max + 1 = 88

[illegible][illegible][illegible][illegible]

- ii. Write algorithms, pseudo-codes, and source code for **selection** sort and **bucket** sort.

### Selection Sort

#### Algorithm

- set the MIN to the 0<sup>th</sup> location.
- Then search the minimum element in the unsorted array.
- Swap the found minimum element with value at MIN.
- Increment MIN by 1 (to the next element).
- Repeat this until the whole list is sorted.

#### Pseudocode

```
function selection sort
    list = array
    n = size of list

    for i=1 to n-1
        //setting current element as minimum
        min = i

        //checking for the minimum element
        for j=i+1 to n
            if list[j] < list[min] then
                min = j;
            end if
        end for

        //swapping the min with current element
        if min != i then
            swap list[min] and list[i]
        end if
    end for
end
```

### Source code

```
package com.selectionsort;

import java.util.Arrays;

public class Main {

    public static void main(String[] args) {
        int[] arr = {87, 35, 28, 26, 9, 12, 7};
        System.out.println("Unsorted Array: \t" + Arrays.toString(arr));

        selectionSort(arr);
        System.out.println("Sorted Array: \t" + Arrays.toString(arr));
    }

    public static void selectionSort(int[] arr) {
        int n = arr.length;
        // one by one move boundary of unsorted subarray
        for (int i = 0; i < n - 1; i++) {
            // find the minimum element in unsorted array
            int min = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[min]) {
                    min = j;
                }
            }
            // swapping the found min value with the first element
            int temp = arr[min];
            arr[min] = arr[i];
            arr[i] = temp;
        }
    }
}
```

### Bucket Sort

#### Algorithm

- Find the maximum number in the array
- Create a new array with size maximum +1
- Iterate over the array, place n at nth position in the second array

### Pseudocode

```
function bucket sort (arr)
    n = arr.length
    B = new array
    for i=1 to n-1
        B[i]=0;
    for i=1 to n
        B[|nA[i]|] = A[i];
    for i=0 to n-1
        sort list B[i] using insertion sort concatenate the lists B[0], B[1], . . . , B[n - 1]
    return B
end
```

### Source code

```
package com.bucketSort;
import java.util.*;
public class Main {
    public static void main(String[] args) {
        int[] arr = {87, 35, 28, 26, 9, 12, 7};
        System.out.println("Unsorted Array:\t"+Arrays.toString(arr)+"\n");
        bucketSort(arr);
    }
    public static int[] bucketSort(int[] arr) {
        //finding the maximum value of the array
        int max = 0;
        for (int i=0; i<arr.length; i++) {
            if(arr[i] > max) {
                max = arr[i];
            }
        }
        //creating a new array
        int[] sortedArr = new int[max+1];
        //storing n at nth position in the new array
        for(int currentVal : arr) {
            sortedArr[currentVal] = currentVal;
        }
        System.out.println("Sorted Array:\t"+Arrays.toString(sortedArr)+"\n");
        return sortedArr;
    }
}
```

- iii. Test your programs with [20, 46, 22, 19, 6, 42, 14, 5, 48, 47, 17, 39, 51, 7, 2] array.

#### Selection Sort

```
package com.selectionsort;

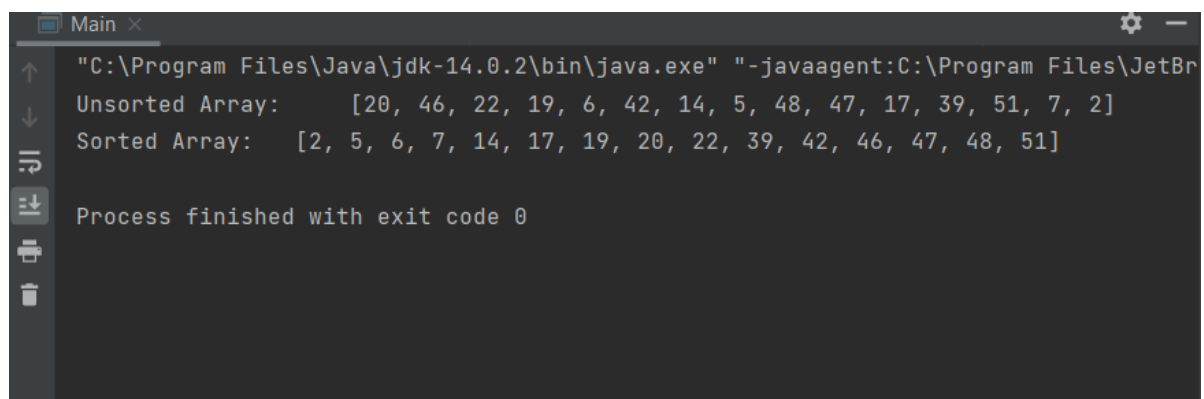
import java.util.Arrays;

public class Main {

    public static void main(String[] args) {
        int[] arr = {20, 46, 22, 19, 6, 42, 14, 5, 48, 47, 17, 39, 51, 7, 2};
        System.out.println("Unsorted Array: \t" + Arrays.toString(arr));

        selectionSort(arr);
        System.out.println("Sorted Array: \t" + Arrays.toString(arr));
    }

    public static void selectionSort(int[] arr) {
        int n = arr.length;
        // one by one move boundary of unsorted subarray
        for (int i = 0; i < n - 1; i++) {
            // find the minimum element in unsorted array
            int min = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[min]) {
                    min = j;
                }
            }
            // swapping the found min value with the first element
            int temp = arr[min];
            arr[min] = arr[i];
            arr[i] = temp;
        }
    }
}
```



```
Main x
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBr
Unsorted Array:      [20, 46, 22, 19, 6, 42, 14, 5, 48, 47, 17, 39, 51, 7, 2]
Sorted Array:       [2, 5, 6, 7, 14, 17, 19, 20, 22, 39, 42, 46, 47, 48, 51]
Process finished with exit code 0
```

## Bucket Sort

```
package com.bucketSortdup;

import java.util.*;

public class Main {

    public static void main(String[] args) {
        int[] arr = {20, 46, 22, 19, 6, 42, 14, 5, 48, 47, 17, 39, 51, 7, 2};
        System.out.println("Unsorted Array:\t"+Arrays.toString(arr)+"\n");

        bucketSort(arr);
    }

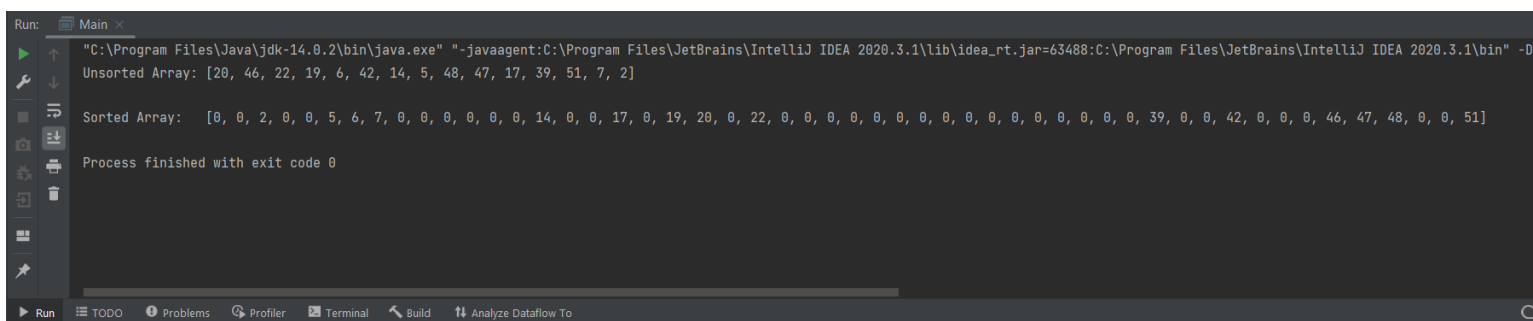
    public static int[] bucketSort(int[] arr) {

        //finding the maximum value of the array
        int max = 0;
        for (int i=0; i<arr.length; i++) {
            if(arr[i] > max) {
                max = arr[i];
            }
        }

        //creating a new array
        int[] sortedArr = new int[max+1];
        //storing n at nth position in the new array
        for(int currentVal : arr) {
            sortedArr[currentVal] = currentVal;
        }

        System.out.println("Sorted Array:\t"+Arrays.toString(sortedArr));

        return sortedArr;
    }
}
```



The screenshot shows the Run window of IntelliJ IDEA. The top bar indicates the file 'Main' is open. The Run configuration shows the command: `"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2020.3.1\lib\idea_rt.jar=63488:C:\Program Files\JetBrains\IntelliJ IDEA 2020.3.1\bin" -D`. The output console displays the following:

```
Unsorted Array: [20, 46, 22, 19, 6, 42, 14, 5, 48, 47, 17, 39, 51, 7, 2]

Sorted Array:  [0, 0, 2, 0, 0, 5, 6, 7, 0, 0, 0, 0, 0, 0, 14, 0, 0, 17, 0, 19, 20, 0, 22, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 39, 0, 0, 42, 0, 0, 0, 46, 47, 48, 0, 0, 51]

Process finished with exit code 0
```

The bottom status bar shows the 'Run' button and other IDE tools like 'TODO', 'Problems', 'Profiler', 'Terminal', 'Build', and 'Analyze Dataflow To'.

iv. Discuss the advantages, drawbacks, and limits of bucket sort.

Advantages

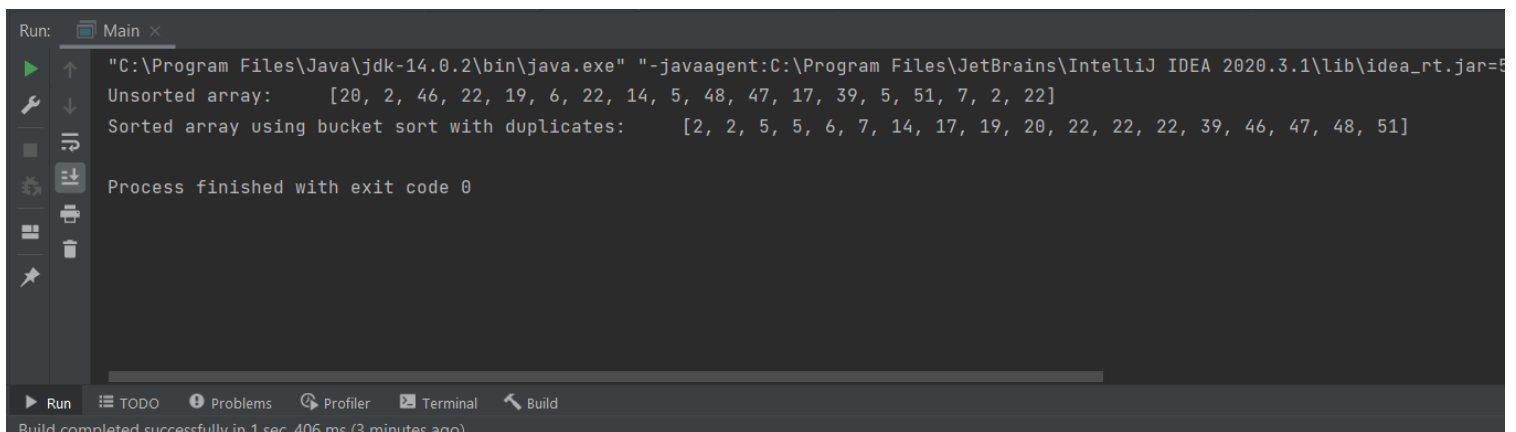
- It is faster than many other sorting algorithms and stable.
- The number of comparisons that needed to be done is less.

Drawbacks and limits of bucket sort

- Duplicates does not work in this sorting algorithm
- Space and memory wastage. This happen due to elements in the array are placed at the nth position in the second array.
- Only works with limited datasets. There's no place to put negative values, fractions, decimals in the array. This only works with 0 and positive numbers only.
- Bucket sort doesn't work well with characters as well, eg: ant, ball, bus. But works well with single characters, such as, a, b, c, d, e, f, etc.

- v. Improve your program to support duplicates for the bucket sort algorithm.
- vi. Validate your program with [20, 2, 46, 22, 19, 6, 22, 14, 5, 48, 47, 17, 39, 5, 51, 7, 2, 22] array.

```
package com.bucketSortDup;  
  
import java.util.Arrays;  
  
public class Main {  
    public static void main(String[] args) {  
        int[] array = {20, 2, 46, 22, 19, 6, 22, 14, 5, 48, 47, 17, 39, 5, 51, 7,  
2, 22};  
        System.out.println("Unsorted array: \t" + Arrays.toString(array));  
        bucketSort(array);  
        System.out.println("Sorted array using bucket sort with duplicates: \t" +  
Arrays.toString(array));  
    }  
  
    public static void bucketSort(int[] arr) {  
        //finding the maximum value  
        int max = arr[0];  
        for (int i = 1; i < arr.length; i++) {  
            if (max < arr[i]) {  
                max = arr[i];  
            }  
        }  
        //creating a new array  
        int[] list = new int[max + 1];  
        for (int i = 0; i < list.length; i++) {  
            list[i] = 0;  
        }  
        for (int i = 0; i < arr.length; i++) {  
            list[arr[i]]++;  
        }  
  
        int index = 0;  
        for (int i = 0; i < list.length; i++) {  
            for (int j = 0; j < list[i]; j++) {  
                arr[index++] = i;  
            }  
        }  
    }  
}
```



```
Run: Main x  
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2020.3.1\lib\idea_rt.jar=5  
Unsorted array:      [20, 2, 46, 22, 19, 6, 22, 14, 5, 48, 47, 17, 39, 5, 51, 7, 2, 22]  
Sorted array using bucket sort with duplicates:      [2, 2, 5, 5, 6, 7, 14, 17, 19, 20, 22, 22, 22, 39, 46, 47, 48, 51]  
  
Process finished with exit code 0  
  
Run  |  TODO  |  Problems  |  Profiler  |  Terminal  |  Build  
Build completed successfully in 1 sec, 406 ms (3 minutes ago)
```



# ALGORITHMS & COMPLEXITY

CS203.3

Dr. Rasika Ranaweera

## Lab Exercise III

**Type:** Individual & Mandatory

**Duration:** 2 Hours (+ Homework)

## Lab Exercise (Sorting Algorithms)

### Important:

- ✓ Objective is to understand, implement, and analyze familiar algorithms.
- ✓ Save your programs under **home/cs203.3/** directory in your computer.
- ✓ Write your name, ID, course code as comments on top of the source files.
- ✓ Learn from others, read books, or Google but DO NOT COPY 1-to-1.

### Assignment:

1. Both “Ternary Search” and “Jump Search” algorithms are extensions to basic linear and binary search algorithms.
  - a. Illustrate both of them with help of examples

#### Ternary Search

To do the ternary search, first the array has to be sorted.

Int[] arr = {2, 5, 5, 7, 10, 10, 12, 15, 18, 20, 21, 24, 29, 34, 36}

2	5	5	7	10	10	12	15	18	20	21	24	29	34	36
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Value to search, sk = 18

Initially,

$$\text{mid1} = (\text{end} - \text{start})/3 + \text{start} = (14 - 0)/3 + 0 = 14/3 = 4$$

$$\text{mid2} = 2 * (\text{end} - \text{start})/3 + \text{start} = 2 * (14 - 0)/3 + 0 = 28/3 = 9$$

start = 0				mid1 = 4					mid2 = 9					end = 14
↓				↓					↓					↓
2	5	5	7	10	10	12	15	18	20	21	24	29	34	36
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

arr[mid1] != sk  
and, arr[mid1] < sk

arr[mid2] != sk  
and, arr[mid2] > sk

sk lies in the interval (mid1,mid2)

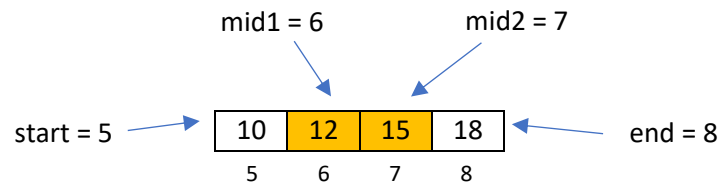
Therefore,

start = mid1 + 1 = 5

end = mid2 - 1 = 8

mid1 = (end - start)/3 + start = (8 - 5)/3 + 5 = 1 + 5 = 6

mid2 = 2 \* (end - start)/3 + start = 2 \* (8 - 5)/3 + 5 = 2 + 5 = 7



arr[mid1] != sk

and, arr[mid1] < sk

arr[mid2] != sk

and, arr[mid2] < sk

sk lies in the interval (mid2,end]

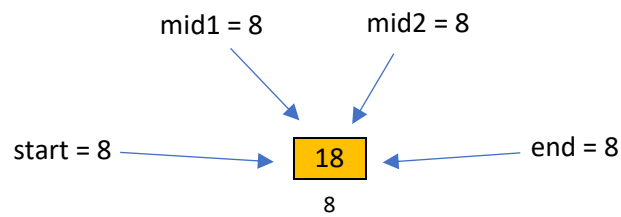
Therefore,

start = mid2 + 1 = 8

end = 8

mid1 = 8

mid2 = 8



arr[mid1] = sk

arr[8] = 18

## Jump Search

Date...../...../.....  
Richard

No:.....

### Jump Search

To do the jump search first the array has to be sorted.

$arr[] = \{0, 1, 1, 2, 3, 6, 9, 9, 11, 14, 19, 22, 24, 26, 29, 31\}$

Value to search,  $x = 11$

Block size,  $m = \sqrt{n} = \sqrt{16} = 4$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	2	3	6	9	9	11	14	19	22	24	26	29	31

$arr[0] \neq x$

$arr[0] < x$ , skip to next block.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	2	3	6	9	9	11	14	19	22	24	26	29	31



$arr[3] \neq x$

$arr[3] < x$ , skip Jump.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	2	3	6	9	9	11	14	19	22	24	26	29	31

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	2	3	6	9	9	11	14	19	22	24	26	29	31

$arr[3] \neq x$

$arr[3] < x$ , skip Jump.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	2	3	6	9	9	11	14	19	22	24	26	29	31

$arr[6] \neq x$

$arr[6] < x$ , Jump

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	2	3	6	9	9	11	14	19	22	24	26	29	31

$arr[9] \neq x$

$arr[9] > x$ , from ~~array~~  $arr[9]$  perform linear search backwards.

Linear Search

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	2	3	6	9	9	11	14	19	22	24	26	29	31

$arr[8] = x //$

$arr[8] = x //$



b. Compute the complexity of the algorithms

Ternary Search

Ternary Search

$T(N)$   $\rightarrow$  Time taken by ternary search for  $N$  elements.

$T(N/3)$   $\rightarrow$  Time taken by ternary search for  $N/3$  elements.

$$T(N) = c + T(N/3) \text{ --- (1)}$$

$$T(N/3) = c + T(N/9) \text{ --- (2)}$$

Substitute (2) in (1);

$$T(N) = 2c + T(N/9) \text{ --- (3)}$$

$$T(N/9) = c + T(N/27) \text{ --- (4)}$$

Substitute (4) in (3);

$$T(N) = 3c + T(N/27) \text{ --- (5)}$$

This can continue like this.

Pattern identified;

$$T(N) = T(N/3^x) + xc \text{ --- (6)}$$

At some point as  $N/3^x$  diminishes, we reach only 1 element in the array. i.e.,

~~$$T(N) = P$$~~

$$T(N/3^x) = T(1)$$

$$\Rightarrow \frac{N}{3^x} = 1$$



Substitute (4) in (3);

$$T(N) = 3c + T(N/3) \text{ --- (5)}$$

This can continue like this.

Pattern identified;

$$T(N) = T(N/3^x) + xc \text{ --- (6)}$$

At some point as  $N/3^x$  diminishes, we reach only 1 element in the array. i.e.,

~~$$T(N) = T(N/3^x) + xc$$~~

$$T(N/3^x) = T(1)$$

$$\Rightarrow \frac{N}{3^x} = 1$$

$$N = 3^x$$

Take  $\log_3$  in both sides,

$$\log_3 N = \log_3 3^x$$

$$\log_3 N = x \text{ --- (7)}$$

Substitute (7) in (6).

$$T(N) = T\left(\frac{N}{3^{\log_3 N}}\right) + c \log_3 N$$

~~also~~ 
$$T(N) = T\left(\frac{N}{N}\right) + c \log_3 N$$

$$T(N) = T(1) + c \log_3 N$$

$$T(N) = k + c \log_3 N$$

Next we find big Oh,

$$T(N) = \log_3 N$$

$$T(N) = O(\log_3 N)$$

Ternary search is  $O(\log_3 N)$  //



## Jump Search

```
public static int JumpSearch (int[] arr, int target) {  
    int n = arr.length; ---- 1  
    int m = (int) Math.floor (Math.sqrt (n)); ---- 1  
  
    int prev = 0; ---- 1  
    while (arr[Math.min (m, n) - 1] < target) { ---- 1  
        prev = m; ----  $\sqrt{n}$   
        m += (int) Math.floor (Math.sqrt (n)); ----  $\sqrt{n}$   
  
        if (prev >= n) { ----  $\sqrt{n}$   
            return -1;  
        }  
    }  
  
    while (arr[prev] < target) { ---- 1  
        prev++; ---- k  
  
        if (prev == Math.min (m, n)) { ---- k  
            return -1;  
        }  
  
        if (arr[prev] == target) { ---- k  
            return prev;  
        }  
    }  
    return -1; ---- 1  
}
```

$$\therefore 3\sqrt{n} + 3k + 6 = O(\sqrt{n}) //$$



- c. Compare (look for similarities) and contrast (look for differences) the above two algorithms

#### Similarities

- Both are searching algorithms
- Elements of the array must be sorted first in order to search.

#### Differences

Ternary Search	Jump Search
<ul style="list-style-type: none"><li>• Divide the array into 3 parts by taking mid1 and mid2 to find the target element. This will go on recursively until mid1 or mid2 will match with the search element.</li><li>• Has a higher complexity than jump search.</li><li>• Divide and conquer approach</li><li>• Comparatively slow than jump search</li></ul>	<ul style="list-style-type: none"><li>• Start searching from the first element of the array and check only fewer elements by jumping ahead by fixed steps (<math>\sqrt{n}</math>) rather than searching all elements.</li><li>• Complexity is less than ternary search.</li><li>• Iterative approach</li><li>• Faster than ternary search as it tries to reduce the number of comparisons.</li></ul>

2. Identify the pros and cons of the four searching algorithms we discussed.

### Linear Search

#### Pros

- Time taking to search for elements in small to medium lists are fast.
- The lists are no need to be sorted to perform linear search.
- Linear search is not affected by insertion and deletions. This is because the linear search doesn't need to be sorted so additional elements can be added and deleted.

#### Cons

- When the array or the list is large and since it is unsorted, it will take time to search for the targeted element. The worst case will be to search through the whole large list and find the targeted element at the end, or not finding the targeted element in the list.
- Has a greater time complexity  $O(n)$  compared to other searching algorithms.

### Binary Search

#### Pros

- Best searching algorithm for large arrays as number of comparisons are less. It eliminates half of the array in every iteration after each comparison.
- It narrow downs the search.
- Much quicker as in every step data need to be searched become half.

#### Cons

- It is more complicated than linear search.
- Work quite slow for arrays of small size.
- Only works for sorted arrays.
- This algorithm is not suitable if elements are constantly added to the array as all the elements in the array should be reordered after every insertion and deletion.

### Ternary Search

#### Pros

- During each comparison 66% of the elements are eliminated from the array.
- Ternary Search can solve all the problems that are solvable using binary search, but binary search can't solve all the problems that are solvable using Ternary search.
- Can use to find maximum and minimum of a function.

### Cons

- Only works for sorted array.
- Ternary search does more comparisons than binary search in worst case.
- Slower compared to binary search and requires more steps to narrow down the search.

## Jump Search

### Pros

- It is faster than linear search as linear search complexity is  $O(n)$  and Jump Search complexity is  $O(\sqrt{n})$ .
- Since it is skipping elements while searching, it reduces the time to search and more efficient than linear search.

### Cons

- Only works for sorted input arrays.
- Slower than binary search.
- Implementation of this algorithm is quite difficult than the binary search.

3. Guess in what occasions these algorithms can be optimal? Provide such occasions for each and every algorithms.

#### **Linear Search**

- Optimal when list of items in a list is small and unsorted, the best algorithm is linear search.

#### **Binary Search**

- In a library books are arranged in some kind of an order, so instead of going through every book (linear search) we can use binary search to find the book.
- In a University, the documents of students are stored in an order, mostly according to the student ID. To access the data best way is to simply use binary search rather than linear search where you have to go through each and every student ID.
- Debugging a code snippet that is somewhat linear. If the code has many steps mostly executed in a sequence and there's a bug, you can isolate the bug by locating the first step where the code produces results.
- Determining the size of your cache for a serving system or deciding on the TTL for the cache.

#### **Ternary Search**

- It's used to find the maxima/ minima of a concave/ convex graph sketched out by a function on multiple inputs.
- This is used in unimodal functions to calculate the function's maximum and minimum values.

#### **Jump Search**

- Best to find values out of a large sorted list of elements, as it needs less comparisons.
- When the binary search is costly best optional algorithm to use is Jump search.

- When jumping back is much slower than jumping forward, jump search comes in handy. This method should be used if jumping back takes substantially longer than jumping forward.

# ALGORITHMS & COMPLEXITY

CS203.3

Dr. Rasika Ranaweera

## Lab Exercise V 2021 Spring

**Type:** Individual & Mandatory

**Duration:** 1 Hour (+ Homework)

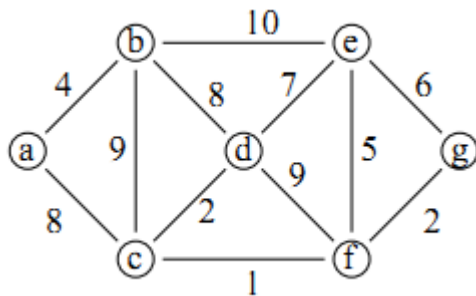
## Lab Exercise (Algorithm Revision)

### Important:

- ✓ Objective is to understand, implement, and analyze new algorithms.
- ✓ Bring your answer sheets (on top write your name, ID, course code) to the next class, after testing your programs on a computer.
- ✓ Save your programs under **home/cs203.3/** directory in your computer (You will be asked to archive/zip "cs203.3" directory before final exam).
- ✓ ONLY **hand-written** answers are permitted.
- ✓ Learn from others, read books, or Google but DO NOT COPY 1-to-1.

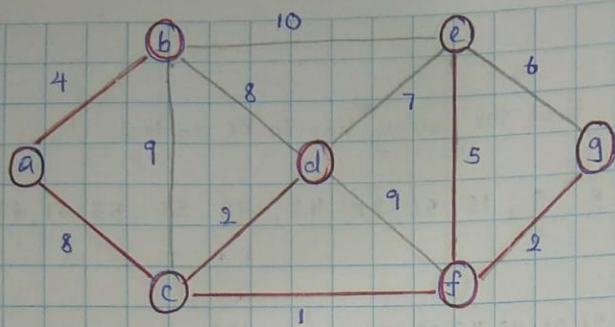
### Assignment:

1. In computer science, Prim's algorithm is usually a greedy algorithm that finds a minimum spanning tree for a **weighted** undirected **graph**.



Source: <http://www.myassignmenthelp.net/prims-algorithm-example>

- i. Draw the minimum spanning tree (MST).



Starting point = ~~d~~ d

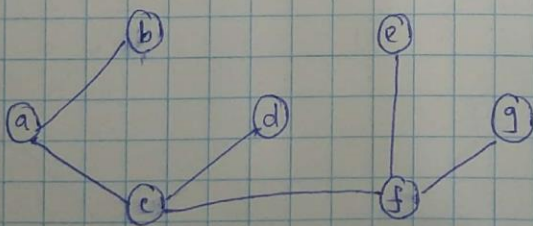
	K	$d_v$	$P_v$
a	T	8	c
b	T	8, 4	d, a
c	T	2	d
d	T	0	
e	T	7, 5	d, f
f	T	9, 1	d, c
g	T	2	f

~~$V = \{a, b, c, d\}$~~

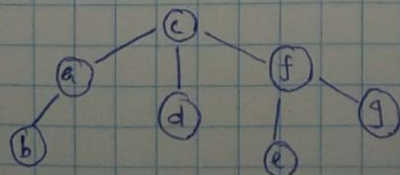
$S = \{d, c, f, g, e, a, b\}$

$$\begin{aligned}\sum d_v &= 8 + 4 + 2 + 0 + 5 + 1 + 2 \\ &= 22 //\end{aligned}$$

Minimum spanning Tree (MST)



Converting MST to a tree.





ii. What is the cost of the MST?

$$8 + 4 + 2 + 0 + 5 + 1 + 2 = 22$$

2. Solve the above graph using Kruskal's algorithm.

Date: .....  
Richard

Edges	$d_v$	
(c,f)	1	✓
(f,g)	2	✓
(d,c)	2	✓
(a,b)	4	✓
(e,f)	5	✓
(e,g)	6	×
(e,d)	7	×
(a,c)	8	✓
(b,d)	8	
(f,d)	9	
(b,c)	9	
(b,e)	10	

$V = \{a, b, c, d, e, f, g\}$   
 $|V| = 7$   
 $|V| - 1 = 6$

Total cost =  $\sum d_v$   
 $= 1 + 2 + 2 + 4 + 5 + 8$   
 $= 22 //$

MST

### 3. Compare and contrast both Prim's and Kruskal's algorithms.

#### Similarities

- Both algorithms are used to find Minimum Spanning Tree (MST).
- Greedy approach is used in both algorithms.

#### Differences

Prim's Algorithm	Kruskal's Algorithm
<ul style="list-style-type: none"><li>• Starting point is an arbitrary vertex.</li><li>• Selected the nodes/ vertices.</li><li>• Here we make sure the nodes are not in the current MST.</li></ul>	<ul style="list-style-type: none"><li>• Starting point is minimum cost edge.</li><li>• Selected the edges.</li><li>• Here we are checking for cycles, if they create a cycle we are not accepting them.</li></ul>