DATA STRUCTURES

gobi.r@christuniversity.in

Agenda

- Insertion Sort, Selection Sort, Merge-Sort, Quick Sort, Heap Sort,
- Linear & Binary Search,
- Hashing, Chaining,
- String matching algorithms: Knuth-Morris- Pratt algorithm.

Todays Agenda

Searching

- *Linear*
- *Binary Search*

Motivation

- It would be an interesting statistics - Pre computer age generations
  - *To have a order*

- *Colossal waste*
  - Sorting and Searching
  - Things are kept properly everything is easy
  - Think if the Dictionary is unordered
  - Google index search - AJAX

To Define

- Searching is an operation which finds the location of a given elements in a list.

- Successful / Unsuccessful
  - *Found or not found*

Types

Linear Search

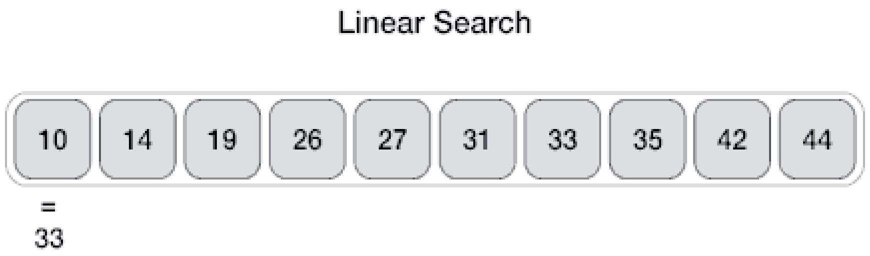
Binary Search

Interpolation Search

Linear Search

- Linear search is a very simple search algorithm.
- In this type of search, a sequential search is made over all items one by one.
- Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection.
- It works on Sorted or unsorted list

Linear Search



The performance of linear search is o(n)

Linear Search

```
int search(int array[], int n, int x)
{
	// Going through array sequentially
	for (int i = 0; i < n; i++)
		if (array[i] == x)
			return i;
	return -1;
}
```

Binary Search iterative

```
do until the pointers low and high meet each other.
    mid = (low + high)/2
    if (x == arr[mid])
        return mid
    else if (x > arr[mid]) // x is on the right side
        low = mid + 1
    else                   // x is on the left side
        high = mid - 1
```

Binary Search recursive

```
binarySearch(arr, x, low, high)
    if low > high
        return False
    else
        mid = (low + high) / 2
        if x == arr[mid]
            return mid
        else if x > arr[mid]        // x is on the right side
            return binarySearch(arr, x, mid + 1, high)
        else                        // x is on the right side
            return binarySearch(arr, x, low, mid - 1)
```

Random number generator

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int c, n;
    printf("Ten random numbers in [1,100]\n");

    for (c = 1; c <= 10; c++)
    {
        n = rand()%100 + 1;
        printf("%d\n", n);
    }

    return 0;
}
```

D:\c-example\generate-random-numbers.exe

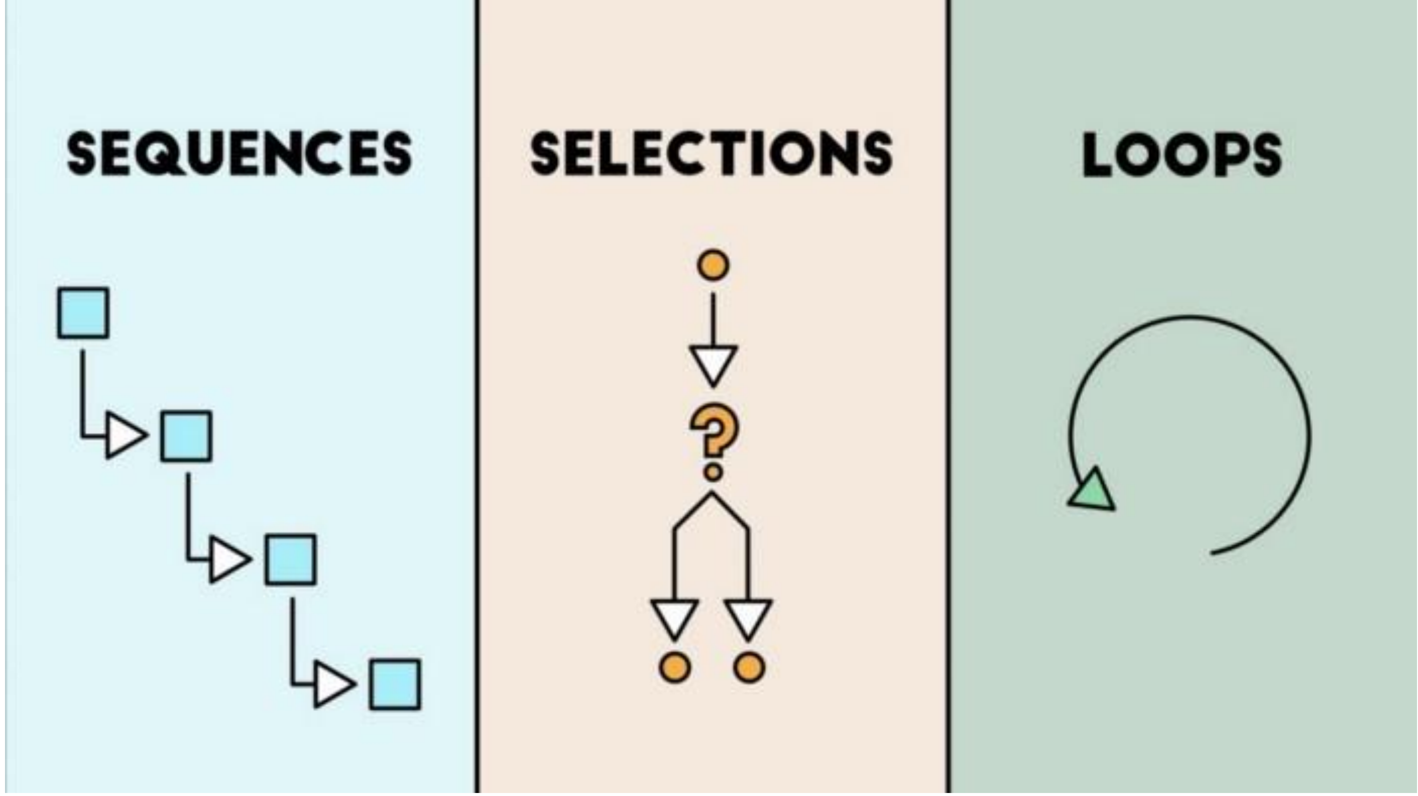
```
Ten random numbers in [1,100]
42
68
35
1
70
25
79
59
63
65

--------------------------------

Process exited with return value 0
Press any key to continue . . .
```

Task to do

- Linear Search
- Use Random function (to generate number)
- Binary Search (Use Sorted numbers)
- Explore Iterative, Recursive methods
- Difference between iteration and recursion
- Use menu driven program for Binary Search


SEQUENCES
SELECTIONS
LOOPS

```python


x = 24
x = 35

y = 9

print(x + y)

```

```python
a = 33
b = 200

if b > a:
    print("b is greater than a")
```

```python
y = [1,2,3,4]

for number in y:
    print(number)
```

```
ListOfPeople = ["Dave", "Phill", "Amanada", "Lucy", "Joe"]

for person in ListOfPeople:
    if len(person) > 5:
        print(person)
```

Pseudocode

- procedure linear_search (list, value)
-     for each item in the list
-         if match item == value
-             return the item's location
-         end if
-     end for
- end procedure

Binary Search

- Binary search is a fast search algorithm with run-time complexity of O(log n).
- This search algorithm works on the principle of divide and conquer.
- For this algorithm to work properly, the data collection should be in the sorted form.

Binary Search

- Very fast & efficient
- Only in Sorted order
- Compare with the centre elements
- Also called as
  - *Half interval search*
  - *Logarithmic Search*
  - *Fast search algorithms*

Binary Search

- Binary search looks for a particular item by comparing the middle most item of the collection.
- If a match occurs, then the index of item is returned. If the middle item is greater than the item, then the item is searched in the sub-array to the left of the middle item.
- Otherwise, the item is searched for in the sub-array to the right of the middle item. This process continues on the sub-array as well until the size of the subarray reduces to zero.

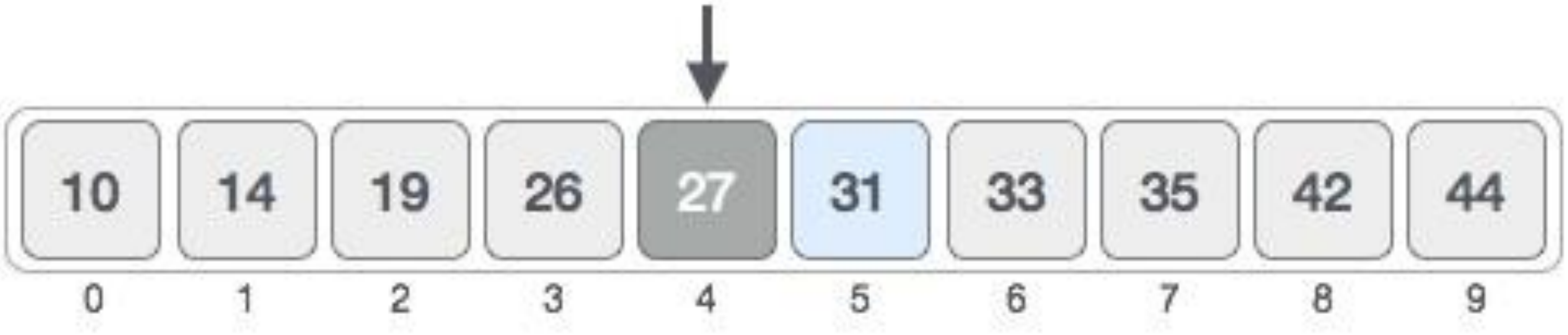
Binary Search

- For a binary search to work, it is mandatory for the target array to be sorted.

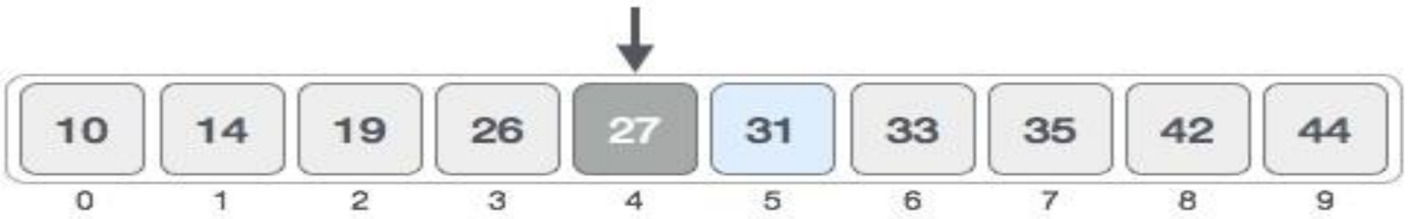
- The following is our sorted array and let us assume that we need to search the location of value 31 using binary search.

How Binary Search Works?

- First, we shall determine half of the array by using this formula
- mid = low + (high - low) / 2
- Here it is, 0 + (9 - 0) / 2 = 4 (integer value of 4.5). So, 4 is the mid of the array.



How Binary Search Works?



- Now we compare the value stored at location 4, with the value being searched, i.e. 31.
- We find that the value at location 4 is 27, which is not a match.
- As the value is greater than 27 and we have a sorted array, so we also know that the target value must be in the upper portion of the array.

How Binary Search Works?

- We change our low to mid + 1 and find the new mid value again.
- low = mid + 1
- mid = low + (high - low) / 2
- Our new mid is 7 now. We compare the value stored at location 7 with our target value 31.

How Binary Search Works?

- The value stored at location 7 is not a match, rather it is more than what we are looking for. So, the value must be in the lower part from this location.
- Hence, we calculate the mid again. This time it is 5.

How Binary Search Works?

- We compare the value stored at location 5 with our target value. We find that it is a match.

- We conclude that the target value 31 is stored at location 5.
- Binary search halves the searchable items and thus reduces the count of comparisons to be made to very less numbers.

Pseudocode

```
Procedure binary_search
   A ← sorted array
   n ← size of array
   x ← value to be searched
   Set lowerBound = 1
   Set upperBound = n
   while x not found
      if upperBound < lowerBound
         EXIT: x does not exists.

      set midPoint = lowerBound + ( upperBound - lowerBound ) / 2

      if A[midPoint] < x
         set lowerBound = midPoint + 1

      if A[midPoint] > x
         set upperBound = midPoint - 1

      if A[midPoint] = x
         EXIT: x found at location midPoint

   end while

end procedure
```

Summary

- Linear
- Binary
- Interpolation search
  - *It is an algorithm for searching*
  - *for a given key in an indexed array that has been ordered by numerical values assigned to the keys (key values).*
  - *It parallels how humans search through a telephone book for a particular name, the key value by which the book's entries are ordered.*