# Assignment No : A1

## 1 Title :

Binary Search using Divide and Conquer.

## 2 Problem Statement :

Using Divide and Conquer Strategies design a function for Binary Search.

## 3 Learning Objectives :

1. To implement binary search using divide and conquer strategy.
2. To learn Scala programming.

## 4 Prerequisites :

1. Basic knowledge of object oriented programming and functional programming.
2. Knowledge of divide and conquer strategy.
3. Knowledge about the binary search algorithm.

## 5 Software and Hardware Requirement :

1. 64 bit Machine i3/i5/i7
2. 64-bit open source Linux OS Fedora 20

3. Python
4. Eclipse

## 6 Concept Related Theory:

### 6.1 Divide and Conquer Algorithm:

- Dynamic Programming is typically applied to optimization problem. In this dynamic programming, the word programming stands for planning and it does not mean by computer programming.
- Dynamic programming is a technique for solving problem with overlapping subproblems.
- In this method each subproblem is solved only once. The result of each subproblem is recorded in the table from which we can obtain a solution to the original problem. For any given problem, we may get any number of solutions we seek for optimal solution (i.e. minimum value or maximum value solution) and such an optimal solution becomes the solution to the given problem.
- Dynamic programming design involves four major steps :
  1. Characterize the structure of optimal solution. That means develop a mathematical notation that can express any solution and subsolution for the given problem.
  2. Recursively define the value of an optimal solution.
  3. By using bottom up technique compute value of optimal solution. For that you have to develop a recurrence relation that a solution to its subsolution using the mathematical notation.
  4. Compute an optimal solution from computed information.

### 6.2 Binary Search Algorithm:

- A binary search or half-interval search algorithm finds the position of a specified value(the input key) within a sorted array.
- In each step, the algorithm compares the input key value with the key value of the middle element of the array. If the keys match, then a matching element has been found so its index, or position, is returned. Otherwise, if the sought key is less than the middle elements key, then the algorithm repeats its action on the sub-array to the left of the middle element or, if the input key is greater, on the sub-array to the right.
- If the remaining array to be searched is reduced to zero, then the key cannot be found in the array and a special Not found indication is returned.

- Every iteration eliminates half of the remaining possibilities. This makes binary searches very efficient - even for large collections.
- Binary search requires a sorted collection. Also, binary searching can only be applied to a collection that allows random access (indexing).

Worst case performance: O(log n) Best case performance: O(1)

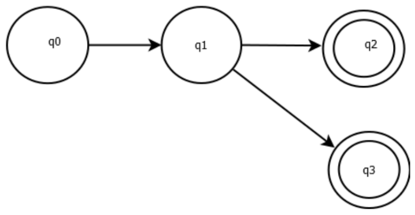
#### 6.2.1 Algorithm:

```
1. Start
2. Create sorted list l
3. Accept key
4. binarysearch(l,key,low,high)
begin
Calculate mid as (low+high)/2
if n < l[mid]
begin
high = mid 1
binarysearch(l,key,low,high)
end
if key > l[mid]
begin
low = mid + 1
binarysearch(l,key,low,high)
end
if key = l[mid]
print found
5. Stop
where, key is the element to be searched and low
and high the lower and upper bounds of the list.
```

## 7 Mathematical Model :

```
Let S be the solution perspective of the system such that,
S ={St ,Et, I, O, DD, NDD, Fme ,Sc,Fc}
where,
St = start state list should be sorted.
I represents set of input
I = {key, sorted array}
key belongs to I+
sorted array belongs to I+
O represents set of output
O = status
where, status=found or not found
Fme =set of functions.
Fme ={f1,f2,f3}
where,
f1= f1 represents the function to read the input.
f2= f2 represents the function to perform binary search.
f3= f3 to display the result.
E t =display result.
DD = Deterministic Data
= list
NDD (Non Deterministic Data)
= the key which may or may not be present in the list
Sc = status=found if key in list else status=not found
Fc = status=found if key not in list else status=not found
```

## 8 State Diagram :



where ,
$q_0$=start state. Accept the key and list.
$q_1$=Perform binary search on the list
$q_2$=Display the status if key found.
$q_3$=Display the status if key not found.

## 9 Program Code:

```
Binary Search function

def binary_search(arr, first, last, key):
if(first <= last):
mid = int((first + last)/2)

if(arr[mid] > key):
mid = binary_search(arr, first,mid-1,key)
elif(arr[mid] < key):
binary_search(arr,mid+1,last,key)
```

```
return mid
else:
return -9999


A1.py
from bin_search import binary_search


print "Enter the number of elements:"
n = int(raw_input())
print "Enter the numbers:"
numbers = []
for i in range (0,n):
numbers.append(int(raw_input()))


numbers.sort()
print "The number entered are:"
print numbers

print "enter the number to search:"
key = int(raw_input())

index = binary_search(numbers,0,n-1,key)
if(index == None):
print "Element not found"
else:
print "Element found at position"
print index
```

## 10 Output:

```
Enter the number of elements:
5
Enter the numbers:
34
67
2
89
29
The number entered are:
[2, 29, 34, 67, 89]
enter the number to search:
```

```
2
Element found at position
0
```

## 11 Conclusion:

We have thus successfully implemented binary search Algorithm using python programming and understood divide and conquer strategy .