**Assignment No.A1**

**1. Problem Statement:**

Binary search using divide and conquer strategies.

**Write down complete problem statement**

**2. Objective:**

* To develop problem solving abilities using Mathematical Modeling.
* To apply algorithmic strategies, Software Engineering and Testing while solving problems.
* To develop time and space efficient algorithms.

**3. Theory:**

**Divide And Conquer Strategy**

The principle behind this strategy is that it is easier to solve small instances of a problem than one large problem than one large complex problem. The divide-and-conquer technique involves in solving a particular computational problem by dividing it into smaller subproblems, soling the problem recursively and then combining the results of all the sub problems to produce the result for the original complex problem ‘P’.

The divide-and-conquer strategy solves a problem by: 1. Breaking it into subproblems that are themselves smaller instances of the same type of problem 2. Recursively solving these subproblems 3. Appropriately combining their answers The real work is done piecemeal, in three different places: in the partitioning of problems into subproblems; at the very tail end of the recursion, when the subproblems are so small that they are solved outright; and in the gluing together of partial answers. These are held together and coordinated by the algorithm’s core recursive structure.

**Binary Search:**

The binary search algorithm begins by comparing the target value

to the value of the middle element of the sorted array. If the target value

is equal to the middle element's value, then the position is returned and

the search is finished. If the target value is less than the middle element's

value, then the search continues on the lower half of the array; or if the

target value is greater than the middle element's value, then the search

continues on the upper half of the array. This process continues, elimi-

nating half of the elements, and comparing the target value to the value

of the middle element of the remaining elements - until the target value is

either found (and its associated element position is returned), or until the

entire array has been searched (and "not found" is returned). The Binary

search algorithm can be applied using two technique i.e

\_ Recursive function

\_ Non Recursive function

(a) Recursive Function: In recursive function the number to be searched

is checked with middle element than if it is the middle element return

the index else pass low,high,number,mid to the function.

(b) Non Recursive Function: In non recursive function the number to

b searched is checked with middle element if its the middle element

return index else with procedural program to find the element.

Time Complexity And Space Complexity:

Worst Case performance O(log n)

Best Case performance O(1)

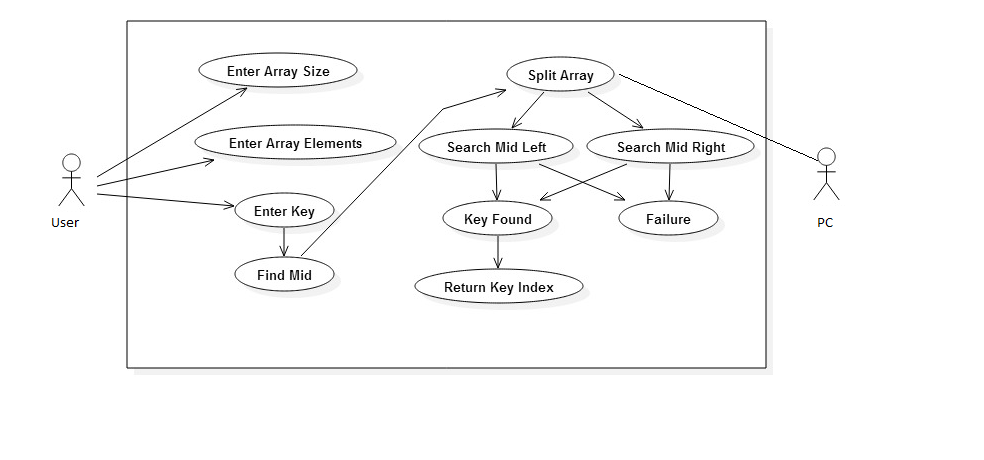
Average case performance O(log n)

Worst Case Space Complexity O(1)

**4.UML Use Case Modeling:**

**USE CASE**

* To model a system the most important aspect is to capture the dynamic behaviour. To clarify a bit in details, dynamic behaviour means the behaviour of the system when it is running /operating.
* So only static behaviour is not sufficient to model a system rather dynamic behaviour is more important than static behaviour. In UML there are five diagrams available to model dynamic nature and use case diagram is one of them. Now as we have to discuss that the use case diagram is dynamic in nature there should be some internal or external factors for making the interaction.
* These internal and external agents are known as actors. So use case diagrams are consists of actors, use cases and their relationships. The diagram is used to model the system/subsystem of an application. A single use case diagram captures a particular functionality of a system.
* So to model the entire system numbers of use case diagrams are used.  
  redraw complete diagram



**5. Algorithm:**

i. Start

ii. Accept size of array i.e. n

iii. Enter the elements in sorted order i.e. a[20]

iv. Accept number to search i.e. no

v. middle,middle=(a[0]+a[n-1])/2

vi. if no==middle then return no.

vii. if no>middle then middle=(a[mid]+a[n-1])/2 and go to step 5

viii. if no<middle then middle=(a[0]+a[mid-1])/2 and go to step 5

**6. Input To Program:**

* Size of array.
* Accept the element.
* The number to search.

**7. Expected Output:**

* Element found at position i.
* Element not found.

**8.Test Cases:**

|  |  |
| --- | --- |
| Scenerio | Expected result |
| Verify that the input field can accept maximum of n number. | The index of array where the number is stored. |

**9.Mathematical Model:**

Let U={S,I,O,F,T}

Where

I={I1,I2}

I1={x1,x2,x3,…,xn} where xn is a natural number

I2={a} where a is a natural number

O={b} where b is a natural number

S=case for success where number to be searched is found

F=case for failure where number to be searched is not found

T={bs()}



**10.Pseudocode**

// initially called with low = 0, high = N - 1

BinarySearch\_Left(A[0..N-1], value, low, high) {

// invariants: value > A[i] for all i < low

value <= A[i] for all i > high

if (high < low)

return low

mid = (low + high) / 2

if (A[mid] >= value)

return BinarySearch\_Left(A, value, low, mid-1)

else

return BinarySearch\_Left(A, value, mid+1, high)

}

**11.Conclusion:**

Hence, we have successfully implemented Binary Search Algorithm using divide

and conquer strategy.

**12.FAQs:**

**1.What do you mean by divide and conquer strategy? How it can be used to increase efficiency of binary search algorithm?**

**Ans.**  Divide and conquer means that we divide the problem into smaller pieces, solve the smaller pieces in some way, and then reassemble the whole problem to get the result.

When we perform a binary search of a list, we first check the middle item. If the item we are searching for is less than the middle item, we can simply perform a binary search of the left half of the original list. Likewise, if the item is greater, we can perform a binary search of the right half. Either way, this is a recursive call to the binary search function passing a smaller list.

**2. What do you mean by object-oriented software design technique? How it could help in designing function for binary search?**

**Ans.**Object-oriented analysis and design (OOAD) is a popular technical approach for analyzing, designing an application, system, or business by applying the object-oriented paradigm and visual modeling throughout the development life cycles to foster better stakeholder communication and product quality.

The primary tasks in object-oriented analysis (OOA) are:

* Find the objects
* Organize the objects
* Describe how the objects interact
* Define the behavior of the objects
* Define the internals of the objects

During object-oriented design (OOD), a developer applies implementation constraints to the conceptual model produced in object-oriented analysis. Such constraints could include the hardware and [software](https://en.wikipedia.org/wiki/Software) platforms, the performance requirements, persistent storage and transaction, usability of the system, and limitations imposed by budgets and time. Concepts in the analysis model which is technology independent, are mapped onto implementing classes and interfaces resulting in a model of the solution domain, i.e., a detailed description of *how* the system is to be built on concrete technologies.

Object-oriented modeling (OOM) is a common approach to modeling applications, systems, and business domains by using the object-oriented paradigm throughout the entire development life cycles. OOM is a main technique heavily used by both OOA and OOD activities in modern software engineering.

Object-oriented modeling typically divides into two aspects of work: the modeling of dynamic behaviors like business processes and use cases, and the modeling of static structures like classes and components. OOA and OOD are the two distinct abstract levels (i.e. the analysis level and the design level) during OOM. The Unified Modeling Language (UML) and SysML are the two popular international standard languages used for object-oriented modeling.

**3. What are the different search algorithms for un-ordered list?**

**Ans.** **Linear Search:**

A linear search is the basic and simple search algorithm. A linear search searches an element or value from an array till the desired element or value is not found and it searches in a sequence order. It compares the element with all the other elements given in the list and if the element is matched it returns the value index else it return -1. Linear Search is applied on the unsorted or unordered list when there are fewer elements in a list.

To search the element 5 it will go step by step in a sequence order.



function **findIndex**(values, target)

{

for(var i = 0; i < values.length; ++i)

{

if (values[i] == target)

{

return i;

}

}

return -1;

}

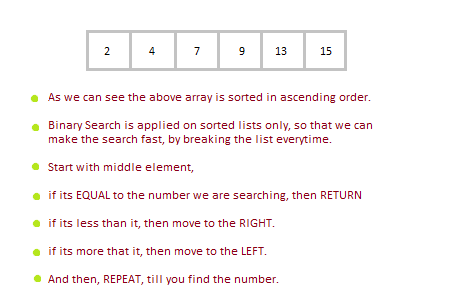
//*call the function findIndex with array and number to be searched*

findIndex([ 8 , 2 , 6 , 3 , 5 ] , 5) ;

**2.Binary Search :**

Binary Search is applied on the sorted array or list. In binary search, we first compare the value with the elements in the middle position of the array. If the value is matched, then we return the value. If the value is less than the middle element, then it must lie in the lower half of the array and if it's greater than the element then it must lie in the upper half of the array. We repeat this procedure on the lower (or upper) half of the array. Binary Search is useful when there are large numbers of elements in an array.

To search an element 13 from the sorted array or list.



function findIndex(values, target)

{

return **binarySearch**(values, target, 0, values.length - 1);

};

function binarySearch(values, target, start, end) {

if (start > end) { return -1; } *//does not exist*

var middle = Math.floor((start + end) / 2);

var value = values[middle];

if (value > target) { return **binarySearch**(values, target, start, middle-1); }

if (value < target) { return **binarySearch**(values, target, middle+1, end); }

return middle; *//found!*

}

findIndex([2, 4, 7, 9, 13, 15], 13);

In the above program logic, we are first comparing the middle number of the list, with the target, if it matches we return. If it doesn't, we see whether the middle number is greater than or smaller than the target.

If the Middle number is greater than the Target, we start the binary search again, but this time on the left half of the list, that is from the start of the list to the middle, not beyond that.

If the Middle number is smaller than the Target, we start the binary search again, but on the right half of the list, that is from the middle of the list to the end of the list.

**4. Compare the linear search and binary search for un-ordered list.(Using the time complexities)**

**Ans.** A **linear search** looks down a list, one item at a time, without jumping. In complexity terms this is an **O(n)** search - the time taken to search the list gets bigger at the same rate as the list does.

A **binary search** is when you start with the middle of a sorted list, and see whether that's greater than or less than the value you're looking for, which determines whether the value is in the first or second half of the list. Jump to the half way through the sub list, and compare again etc. This is pretty much how humans typically look up a word in a dictionary (although we use better heuristics, obviously - if you're looking for "cat" you don't start off at "M"). In complexity terms this is an **O(log n)** search - the number of search operations grows more slowly than the list does, because you're halving the "search space" with each operation.

**CODE:**

alist = []

inp = open ("file","r")

for line in inp.readlines():

for i in line.split():

alist.append(int(i))

print "The entered list is: "

print alist

alist.sort()

print "Sorted list is : ",alist

x=input("Enter the number to be searched :")

def binarySearch(alist, item):

first = 0

last = len(alist)-1

found = False

while first<=last and not found:

midpoint = (first + last)//2

if alist[midpoint] == item:

found = True

else:

if item < alist[midpoint]:

last = midpoint-1

else:

first = midpoint+1

if found == True:

return midpoint

else :

return -1

f=(binarySearch(alist, x))

if f==-1:

print "number not found"

else:

print "number found at postion : ",f+1

**FILE:**

4 2 3 6 1

**OUTPUT**

rajani@rajani-Inspiron-5521:~/Downloads$ python bin.py

The entered list is:

[4, 2, 3, 6, 1]

Sorted list is : [1, 2, 3, 4, 6]

Enter the number to be searched :4

number found at postion : 4

rajani@rajani-Inspiron-5521:~/Downloads$ python bin.py

The entered list is:

[4, 2, 3, 6, 1]

Sorted list is : [1, 2, 3, 4, 6]

Enter the number to be searched :5

number not found

rajani@rajani-Inspiron-5521:~/Downloads$