**ASSIGNMENT NO: A1**

# 1. TITLE

Using Divide and Conquer Strategies and object-oriented software design technique using Modelio to design a software function for Binary Search for an un-ordered data stored in memory. Use necessary USE-CASE diagrams and justify its use with the help of mathematical modelling and related efficiency. Implement the design using Python.

# 2. PREREQUISITES

* 64-bit Fedora or equivalent OS with 64-bit Intel-i5/i7
* Python 2.7

# 3. OBJECTIVE

* To Implements the Ordered search approach for given number..
* Implementation search method.

# 4. MATHEMATICAL MODELS

Let, S be the System Such that,

A={ S, E, I,O, F, DD, NDD, F\_min ,F\_fri, CPU\_Core, Mem\_Shared, success, failure }

Where,

S= Start state,

E= End State,

I= Set of Input

O= Set of Out put

F =Set of Function

DD=Deterministic Data

NDD=Non Deterministic Data

F\_Min=Main Function

F\_Fri= Friend Function CPU\_Core= No of CPU Core.

Mem\_ Shared=Shared Memory.

Function:

1. Splitting Function = This function is used for splitting unsorted list.
2. Sorting Function = This function is used for sorting list.
3. Binary Search = This function apply binary search on sorted list.

Success Case: It is the case when all the inputs are given by system are entered correctly. Failure Case: It is the case when the input does not match the validation Criteria.

**5. THEORY**

# Divide and Conquer

The most well-known algorithm design strategy, Given a function to compute on n inputs, the divide-and-conquer strategy consists of:

1. **Divide**the problem into two or more smaller sub-problems. That is splitting the inputs into k distinct subsets, 1 k  n, yielding k sub-problems.
2. **Conquer**the sub problems by solving them recursively*.*
3. **Combine**the solutions to the sub problems into the solutions for the original problem.
4. if the sub-problems are relatively large, then divide\_Conquer is applied again.
5. if the sub-problems are small, then sub-problems are solved without splitting.

**A typical Divide and Conquer case:**

**A problem of size n**

**Subproblem 2**

**of size n/2**

**Subproblem 1**

**of size n/2**

**A solution of subproblem**

**1**

**A solution of**

**subproblem 2**

**A solution to the original**

**problem**

**Fig. Divide and Conquer Strategy**

# General method of Divide and Conquer algorithm

|  |
| --- |
| Divide\_Conquer(problem P)  {  if Small(P) return S(P); else {  divide P into smaller instances *P*1, *P*2, …, *Pk*, *k*1; Apply Divide Conquer to each of these subproblems ; return  Combine (Divide\_Conque(*P*1), Divide\_Conque (*P*2),…,………Divide\_Conque (*Pk*));  }  } |

# BINARY SEARCH

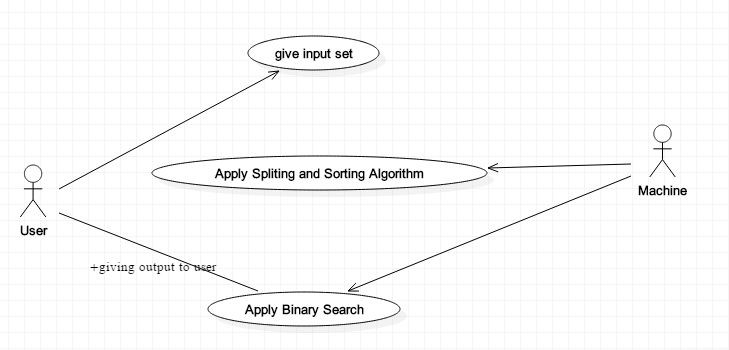
|  |
| --- |
| 1. Algorithm Bin search(a,n,x) 2. // Given an array a[1:n] of elements in non-decreasing 3. //order, n>=0,determine whether ‘x’ is present and 3. // if so, return ‘j’ such that x=a[j]; else return 0. 4. { 5. low:=1; high:=n; 6. while (low<=high) do 7. { 8. mid:=[(low+high)/2]; 9. if (x<a[mid]) then high; 11. else if(x>a[mid]) then low=mid+1; 10. else return mid; 11. } 12. return 0; 13. } |

* Algorithm, describes this binary search method, where Binsrch has 4I/ps a[], I , l & x.
* It is initially invoked as Binsrch (a,1,n,x)
* A non-recursive version of Binsrch is given below.
* This Binsearch has 3 i/ps a,n, & x.
* The while loop continues processing as long as there are more elements left to check.
* At the conclusion of the procedure 0 is returned if x is not present, or ‘j’ is returned, such that a[j]=x.
* We observe that low & high are integer Variables such that each time through the loop either x is found or low is increased by at least one or high is decreased at least one.
* Thus we have 2 sequences of integers approaching each other and eventually low becomes > than high & causes termination in a finite no. of steps if ‘x’ is not present.

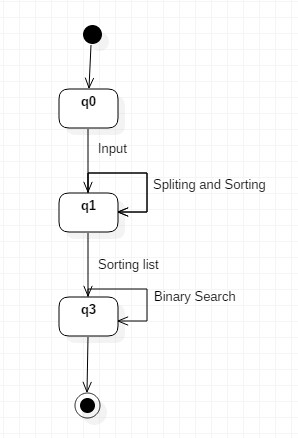
# 6. APPLICATION FLOW

* start with our root/goal node and check current vertex is the goal state
* treat List as stack
* new search states to explore at front of list
* put new states=use heuristics
* leaf node in search List
* Use Backtrack for higher node.

# 7. UML Diagrams



**Fig: Use case Diagram**

 **Fig: State Diagram**

**8. CONCLUSION**

Binary search method using divide and conquer strategy is implemented.

Code

def binarySearch(alist, item):

"""

binartsearch using divide and conqure(non-recursive).

   @param: alist, unsorted list

   @param: item, an element to be searched in alist

   returns: bool, position, the presence of item in alist and position.

"""

pos = 0

first = 0

found = False

alist1=list()

atup = tuple() #temp

adict = dict() #{sortedlist:originalposition}

if isinstance(alist,list):

atup = tuple(alist)

alist1 = list(atup)

alist.sort()

#to store the position of each item in alist into adict

for i in range(len(alist)):

adict[alist[i]] = alist1.index(alist[i])+1

#actual program begins here--..

last = len(alist)-1

    while first<=last and not found:

       midpoint = (first + last)//2

       if alist[midpoint] == item:

           found = True

           pos = adict[item]

       else:

           if item < alist[midpoint]:

               last = midpoint-1

           else:

               first = midpoint+1

#print adict

return found,pos

testlist = [42,72, 2, 11, 55, 32, 76]

print(binarySearch(testlist, 2))

print(binarySearch(testlist, 13))

#output

'''

cipher@blackfury-HP-eNVy:~/be-2$ python binarysearch.py

(True, 3)

(False, 0)

'''