$N - Ary$ Search Time Complexity Analysis

Tarun Kumar
UE143103
Computer Science Engineering
University Institute of Engineering and Technology
Panjab University
Chandigarh, 160030
Email: tarunk.1996@gmail.com

***Abstract*—This practical analyses the N-Ary Search Time Complexity for different values of N. The practical approach used behind this, for locating the Key element in quickest way possible. This practical will deal with the factors that are responsible and affects curve observed in case of N-ary search.**

I. Introduction

N-Ary Search is a technique to search an element in a given sorted list by dividing the list in n different equal parts and search the element in the divided list recursively (or iteratively) until the element is not found or the list gets drained out completely. In this case, if the value of the N is 2 then the searching algorithm used is known as Binary Search Algorithm. The only condition which is necessary before searching for any number in the list of elements that the list must be in sorted order either in increasing or decreasing order. This experiment analyzes the change in the search time taken for a key element to be located in the given list as the number of divisions (i.e. value of N) keeps on increasing at every iteration.

II. Problem

To analyze the search time complexity of N-ary search algorithm and plotting a curve for different values of N.

III. Aim

To write a program which finds the average time taken to search a key element in a given sorted list filled with random numbers. Also, plot a graph demonstrating the total average time taken to search the key element in the sorted list of numbers. Discuss the factors affecting the nature of the curve in detail.

IV. Approach

Define a large array of type integers and populate the array using random function (numbers must fall under some particular range). Initialize a key element with some random number within the same range of numbers. Sort the array filled with numbers using some inbuilt functions or with user-defined function (Note: The time required to sort all the elements of array element must not be included under the search time). Now, note the clock time or assign the current system time to some variable. Call the N-Ary user defined function passing array, its beginning, ending, key element and value of N (i.e total number of divisions to be done in the array) as arguments to the function. Once the element is found return its index value to the parent function or in case of not found condition return -1 to the parent function. Again note down the clock time to evaluate the total time elapsed to locate the key element in the sorted array of numbers. Repeat the same procedure for same values of array and key elements but with different value of N this time. Record the time taken to perform the search procedure on the array elements for different values of N (Number of divisions). To get a precise value, repeat the above steps multiple number of times and write the average recorded time used to search the key element into some file. Based on the given result plot a graph having number of divisions on X-axis and Average time taken on Y-axis of the graph.

V. Experiment

To achieve the goal of the experiment,the program must follow some particular criteria which is discussed in this section in detail. The experimental results obtained in this practical are based on the following approach and implementation. N-Ary search problem can be solved by executing following steps :

1) Declare an array of size 200 and some more variables which will be used in future of type integer.
2) Declare another array of size 100 for storing the average time taken in order to search a key element using divide and conquer strategy for different values of N.
3) Populate the first array with random numbers using rand() function(Make sure to specify a domain for it). It can be done using rand()%domain. Also, initialize the key element with a random number too.
4) Before passing this array to any function. It must fulfill the condition that it is in sorted order. So, it must be passed to some inbuilt sort function or user defined function to sort the array in ascending order.
5) Set the current clock time to a variable using clock() function.(It will be used to calculate the time taken to search key element in sorted list).
6) Call the N-ary user defined function passing all the necessary arguments. The working procedure of the function is explained in the next section.

7) Once the searching process is done. There can be two valid possibilities i.e. either the key element is found or not found. In both the cases the function have accessed the whole list and compared the value of key with partition breaker.
8) Iteratively call the same function having same array and key elements value but with different N value. Repeat these steps for value of N ranging from 3 to 149.
9) Repeat the process from 3 to 8 atleast 10,000 number of times to get a precise value of the total time taken to search a key element for different values of N in divide and search strategy.

The results discussed in the following experiment is performed on a Windows 8.1(x64 bit) based system. To minimize the hindrance due the to heavy background process were terminated. But some small system process were still running in the background during the program execution.

VI. Algorithm for N-ary Search

```
algorithm searching(a[],s,e,d,key)
{
  if(s>=e) then
  {
      if(key==a[s]) then
      {
          return e;
      }
      else
          return -1;
  }
  if(d<1) then
      return -1;
  else
  {
     b=s,pre=b,i=0;
    while(i<d+1) do
    {
      pre=b;
      i++;
      b=((e-s)+1)*i/d;
    }
    if(key==a[b]) then
      return b;
    else
    {
     if(b-pre>d) then
      return searching(a,pre,b-1,d,key);
     else
      return linear(a,pre,b-1,key);
    }
  }
}
```

VII. Results and Graphs

Following are the graphs obtained after performing the experiment as specified above :

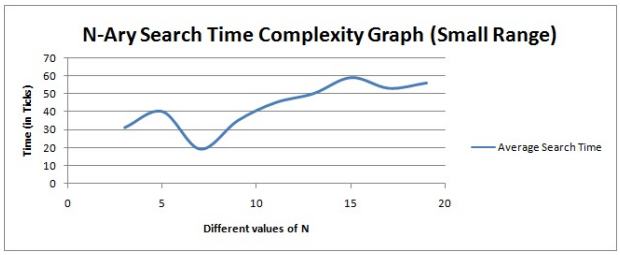

Fig. 1. Search time complexity of N-ary for a small range of numbers

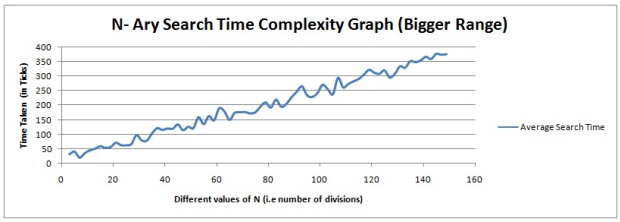

Fig. 2. Search time complexity of N-ary for a bigger range of numbers

VIII. Results Obtained

Value of N	Time taken	Value Of N	Time taken
3	31	77	196
5	40	79	210
7	19	81	192
9	35	83	220
11	45	85	195
13	50	87	205
15	59	89	228
17	53	91	247
19	56	93	266
21	71	95	235
23	62	97	229
25	62	99	242
27	66	101	271
29	97	103	256
31	79	105	238
33	78	107	295
35	103	109	262
37	121	111	274
39	115	113	283
41	120	115	291
43	119	117	306
45	134	119	323
47	114	121	313
49	126	123	309
51	121	125	321
53	159	127	296
55	135	129	309
57	163	131	335
59	148	133	230
61	190	135	353
63	179	137	349
65	150	139	355
67	174	141	368
69	176	143	360
71	177	145	378
73	172	147	375
75	176	149	377

IX. Discussion on the results

In figure 1, it is observed that for N=3 divisions in a sorted array, the time taken to locate the key element in it is 31 Ticks(unit of time) and for N=5 time jumped to 40 which is not appropriate according to the expectation. This anomaly in the behavior of curve could be because of the occurrence of worst cases in case of N=5 which results in the hike observed in the curve. As the value of N or number of partitions increases it is expected to show a decline in the curve because according to the formula:
$T(n) = a * T(n/k) + f(k)$.
But here while searching, only one part of a partition is used for further searching. So, the value of a will be 1 in this case. Hence, the formula that will be used is :
$T(n) = T(n/k) + f(k)$.
Here, k represents number of divisions taking place and $f(k)$ is some function of k (increases as the value of k increases). So, After solving the particular recurrence relation $T(n)$ comes out to be $log_k(n) + f(k)$. So, firstly $log_k(n)$ is dominating in nature compared to $f(k)$. But as value of k increases, a shift is observed and $f(k)$ starts dominating $log_k(n)$.Also, $log_a(x) < log_b(x)$ only when $b < a$. Hence, a minima is observed in the graph were the time complexity is minimum compared to others.
Similarly in the graph a local minima is observed at k=7, where avg time taken to search an element in an array is 19 ticks only. After k=7 the graph slope shifts to positive (i.e increasing nature) because $f(k)$ dominates $log_k(n)$. A periodic increase in time taken is observed in the graph until 23. At k=23 the time taken in ticks is dropped from 71 to 62 instantly. This must have happened due to the background system processes that were running parallel with the program execution or due to the occurrence of worst cases (i.e the key element was found at the center or not found in the list).
Also from figure 2. It's getting clear that the growth of curve is comparable with linear growth function of a curve. Based on the fact, $f(k)$ is predicted to be a linear function. Also, as the value of k reaches to N the time complexity of search is almost equal to linear search algorithm i.e $O(n)$.
In this experiment the size of array is taken to be a constant value 200. In the $N-ary$ recursive function array, key value, its starting, ending element id and k number of divisions is passed. Inside the function when the value of divisions exceeds the total size of partitioned list. A linear search algorithm is called because the time complexity of both cases is same and total number of comparisons that has to be performed is equal to that of number of comparisons perform in linear search.
A global minima is observed at k=7 but at some more minimas 23 and 31 where a certain drop in the graph is seen. Thus, according to experimental results obtained, 7 is global minima where the value of function $T(n) = log_k(n) + f(k)$ is minimum compared to any other value in range of [3,149].

X. Conclusion

- In the experiment N-ary search time complexity, a global manima is observed at 7 where the function $T(n)$ is minimum.
- Other than global minima, a lot of other minimas are also observed and the factors responsible for it could be the system processes or occurrence best cases.
- Other than global minima, a lot of other minimas are also observed and the factors responsible for it could be the system processes or occurrence best cases.
- After certain value the growth of the graph is similar to the graph obtained in linear search having time complexity $O(n)$.
- At the beginning of the curve a certain decrease is observed due to factor $log_k(n)$ in the function. n is taken to be constant value and as k increases, the value of function decreases.

Thus we can achieve better search time complexities as we increase the number of partitions though till a point which gives the best searching time