*IOSR Journal of Computer Engineering (IOSR-JCE)*
*e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 18, Issue 3, Ver. I (May-Jun. 2016), PP 65-68*
*www.iosrjournals.org*

# Dn Searching Technique

Deep Chandra Andola[1], Nitin Deepak[2]
[1,2]*Computer Science & Engineering, Amrapali Group of Institute, Haldwani*[1]

***Abstract:*** *Searching in computer science is the method which finds the required item in the given data. Linear Search is the technique for extracting a particular item in the list that checks each part in series until the preferred element is found or the list is exhausted. The sequence needs not to be in some order. The complexity for the linear search is θ (n).In this paper, the algorithm is discussed takes less time with respect to linear search. Time taken by the DN Search algorithm is equal to time taken by Binary Search Algorithm, but this algorithm also works on unsorted data.*

## I. Introduction To Algorithms

In mathematics, an algorithm is a self-sufficient bit by bit set of operations to be performed. It is a productive method that is defined within a restricted quantity of space and time and in a distinct formal language for calculating a function [1] [2].

In computer system, an algorithm is essentially an occurrence of logic written in software by software developers to be useful for the planned "objective" computer(s) for the target machines to develop output from known input (perhaps null). Moreover, each and every algorithm must suit the following criterion:

- **Input**: there can be one or more quantities which are supplied externally.
- **Output**: Minimum one quantity that is produce.
- **Definiteness**: Each and every order must be understandable, accurate and exact.
- **Finiteness**: If we trace the commands of an algorithm, then each and every case of the algorithm will end after the limited number of steps.
- **Effectiveness**: Each and every order of an algorithm must be suitably basic that it can be standard to be approved by a person by means of only pen and paper. It is not sufficient that each process be exact, but it is also be reasonable.

## II. Understanding Communication Of Time & Space Trade-Off For Algorithms

A problem may have number of solutions. In order to select the most suitable algorithm for a given assignment, you require judging, in how much time a particular result will execute. Or, more precisely, you need to be able to judge how long two solutions will execute, and decide the superior from them. You don't require to know how much time they will take, but you do need some technique to compare algorithms beside one another [1] [4].

Asymptotic complexity is an approach of expressing the key module of the cost of an algorithm, using idealized (not analogous) units of computational task. For example, consider the algorithm for sorting a deck of cards, which proceed by repeatedly searching through the deck for the lowest card. The asymptotic complexity of above algorithm is square of the total number of cards in the deck. Quadratic behaviour of an algorithm is the main term in the complexity formula, for example, if we twice the number of cards in the deck, then the job is roughly quadrupled.

Now let us think how we can compare the complexity of two or more algorithms. Let f (n) is the cost (worst case) of any algorithm, expressed as a function in input size *n*, and g (n) be the cost of other function. Example, for searching algorithms, f (10) and g (10) would be the maximum steps that the algorithms would take on a list of 10 elements. If, for all values of n >= 0, *f (n) <= g (n)*, then the algorithm with complexity function f is strictly quicker. But, in general, our concern for computational cost is for the cases with large input; so the comparison of f (n) and g(n) for small values of n is less significant than the "long term" comparison of f(n) and g(n), for n larger than some threshold.

There are 3 asymptotic notations, which are mostly used to symbolize time complexity of algorithms.

1. **θ Notation*:*** This functions bounds from above and below, so it gives exact asymptotic behaviour. The easy way to get Theta notation of an expression is to drop low order terms and ignore leading constants.

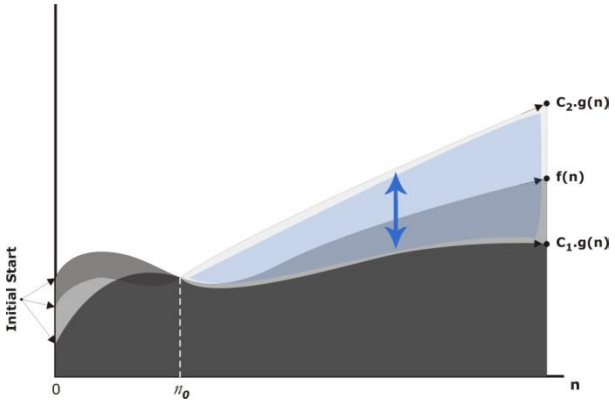


**Figure 1: f(n) = θ (g(n))**

**θ** (g(n)) = {f (n): there subsist positive constant $C_1$, $C_2$ and $n_0$ such that
0 <= $C_1$*g (n) <= f (n) <= $C_2$*g(n) for all n >= $n_0$

2. **Big 'Oh' (O) Notation**: The Big *O* notation explain an upper bound of an algorithm; it bounds a function only from above. For example, let us consider the case of Insertion Sort. It takes linear time and quadratic time in best and worst cases respectively. We can safely say that the time complexity of Insertion sort is O ($n^2$). Note that O ($n^2$) also covers linear time.

The Big O notation is useful when we only have upper bound on time complexity of an algorithm.

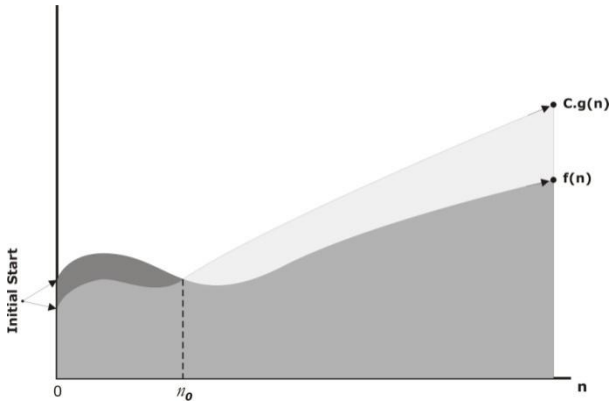


**Figure 2: f(n) = O (g(n))**

**O** (g(n)) = {f (n): there subsist positive constant C and $n_0$ such that
0 <= f (n) <= C*g(n) for all n >= $n_0$

3. **Ω Notation**: As Big O notation provides an asymptotic upper bound, **Ω** notation provides an asymptotic lower bound for a function. **Ω** Notation can be helpful when we have lower bound on time complexity of an algorithm.

Ω *notation* expresses the best case of a function means at least an algorithm runs T times.

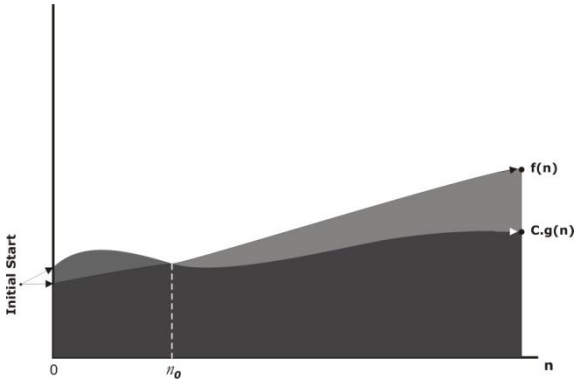


**Figure 3: f(n) = Ω (g(n))**

**Ω** (g (n)) = {f (n): there subsist positive constant c and $n_0$ such that 0 <= C*g (n) <= f (n) for all n >= $n_0$

## III. Linear Search

In computer science, linear search (also known as sequential search) is a technique for finding a particular element in a given list that checks each element in series until the required element is found or the list is exhausted. The record need not be in any order. Sequential search is the simple searching algorithm; it is a case of brute-force search. Its worst case cost is directly proportional to the total number of elements in the given list. Its expected cost is also relative to the number of elements if all elements are searched uniformly.

**Algorithm for Linear Search**

Here '***A***' is the linear array with '***n***' values and an '***element***' is the item that is to be searched in the given array. This algorithm find the location '***loc***' of given item in array *A*, or set loc = 0 if the ***element*** is not found or search is unsuccessful [3].

**linear_search (A, *element*)**

1. [place ***element*** at the end of array, i.e. 'A']

A [n+1] = ***element***

2. [start counter]

loc = 1

3. [explore for ***element***]

Do again while A [loc] =! ***element***
loc = loc + 1
4. [successful]
if loc = n + 1
else set loc = 0
Exit

**Analysis:**

For an array with n items, the best case is when the item is equal to the starting element of the array, in which case, only one comparison is required. The worst case is when the value is not present in the list (or occurs at the end), where n comparisons are needed. Thus, Linear Search worst case complexity is **θ (n)**.

## IV. Binary Search

In computer science, a binary search algorithm finds the location of a particular input value (*key*) inside a given array, which is sorted by key value. For binary search, the given array should be organized in a particular order (ascending or descending order). In every step, the binary search compares the *key* value with the key value of the center element of the given array. If the key matches, then a matching element has been found and its position (or index), is returned. Otherwise, if the *key* is smaller than the centre element's value, then the algorithm repeats its process on the sub-array to the left side of the middle element or, if the *key* is larger, on the sub-array to the right side of the middle element. If the remaining array that is to be searched is vacant, then the *key* value cannot be found in the given array and a special case of "not found" indication is returned.

**Algorithm for Binary Search**

Here '***A***' is ordered array (ascending or descending) with lower bound '***P***' and upper bound '***R***' and '***key***' is an element of information that is to be searched in ***A***. The different variables ***beg***, ***end*** and ***mid*** denote the starting, last and the centre location of the segment of element *A*. This algorithm finds the location ***loc*** of *key* in *A* [3].

**binary search (A, P, R, item)**

1. [initialize section variables]
   Beg = P
   End = R
   Mid = int ((beg + end) / 2)
2. Repeat step 3 & 4
   While (beg <= end) && (A[mid] =! key)
3. if (key < A[mid], then
   End = mid -1
   Else
   Beg = mid + 1
   [if structure terminates]
4. mid = int ((beg + end) / 2)
   [step 2 loop ends]
5. if A[mid] = key
   then loc = mid
   else
   loc = NULL
6. Exit

**Analysis*:***

Log (N) −1 is the expected number of steps in an standard successful cases, and the worst case is log(N), just one more step. If the list is blank, no steps at all are required. Thus, we can say that binary search is a logarithmic algorithm and completes in **θ (log n)** time. In most cases it is significantly quicker than a linear search.

As Linear & Binary Search Techniques are discussed and their average time can be given as:

| SNo | Technique | Data List Format | Average Time |
|---|---|---|---|
| 1. | Linear Search | Any | T (n) = **θ (n)** |
| 2. | Binary Search | Sorted | T (n) = **θ (log n)** |

**Table 1: Comparison between Binary Search & Linear Search**

Since, time taken by the Linear Search is greater than the Binary Search. But the constraint in binary search is that list should be sorted.

## V. DN Search

DN Searching Algorithm is the searching technique. This algorithm is similar to Linear Search algorithm, i.e. it works on unsorted data. But DN searching technique takes less time then Linear Search. Time taken by this algorithm is equal to the time take by the Binary Search Algorithm, other than Binary searching technique should apply on sorted data only, if data or list is in unsorted then Binary searching technique is not valid. So, DN Searching technique can be applied on sorted or unsorted data or list. Algorithm for DN Searching Technique is discussed below:

**Algorithm for DN Algorithm**

Here, '***A***' is the list with '***n***' element

'***num***' item to search in *A*

'***B***' is the array of length 2

*B [1]* store '***1***' for TRUE and

*B [2]* store position of the result

'***search()***' function is used to compare

```
DN_Search (A, num, n)
Line1:  for (i = 1 to n-1) {
Line2:  sum = A[i] + A[i+1];
Line3:  if ((sum – num) >= 0)
Line4:  B = search (A, i, i+1, num);
Line5:  if (B[1] = = 1)
Line6:  break;
Line7:  i = i + 2;
Line8:  }
Line9:  if (B[1] != 1 && i <= len) {
Line10: if (num = =A[i])
Line11: B[1]=1;
Line11: B[2]=i;
Line12: }
Line13: if (B[1] = = 1) {
Line14: pos = B[2];
Line15: print "element is at A[pos] position";          // successful
Line16: } else
Line17: print "NOT FOUND";                              // unsuccessful
search (A, strt, end, num)
Line1s: x[1]=0; x[2]=0;
Line2s: for (k = strt to end){
Line3s: if (num = = A[i]){
Line4s: x[1]=1;   x[2]=i;
Line5s: }
Line6s: }
Line7s: return x;
```

## VI. Result

As all three searching techniques are discussed and their average time can be given as:

| SNo | Technique | Data List Format | Average Time |
|---|---|---|---|
| 1. | Linear Search | Any | T (n) = **θ (n)** |
| 2. | Binary Search | Sorted | T (n) = **θ (log n)** |
| 3. | DN Search | Any | T (n) = **θ (log n)** |

**Table 2: Comparison between Linear, Binary Search & DN Search Technique**

Since, average time taken by DN and Binary Search Technique are same. But DN Searching Technique is more powerful because it can be implemented on unsorted data or list, while Binary Search needs only sorted list.

## VII. Conclusion

In this paper, Linear and Binary Search techniques were discussed with their average time (in Table1), in which binary search was superior to linear search with lesser time. In Table2, when new search technique (DN Search) was compared with other two, time taken by the binary and DN search was equal and better than that of linear search. But the limitation of binary search was dominated by DN search, that is, DN search technique can be applied on any type of list (sorted or unsorted).

Thus, DN Search Technique can be proved as more powerful technique than Linear & Binary Search.

## References

[1] Thomas H Cormen, Charles H Leiserson, Ronald L Rivest, Clifford Stein, "Introduction to Algorithm", 3rd Edition, Prentice Hall of India Pvt Ltd., Chapter 1 and Chapter 3.
[2] Ellis Horowitz, Suraj Sahni, Sanguthevar Rajasekaran, "Fundament of Computer Algorithms", Galgotia Publication, Chapter 1.3.
[3] Seymour Lipshutz, Schaum's Outline "Data Structures", Tata McGraw Hill, Chapter 4.7 and Chapter 4.8.
[4] Seymour Lipshutz, Marc Lars Lipson, Varsha H Patil, Schaum's Outline "Discrete Structure", Third Edition, Tata McGraw Hill, Chapter 3.9