ISSN: 2348 - 2117

International Journal of Engineering Technology and Computer Research (IJETCR)
Available Online at www.ijetcr.org
Volume 4; Issue 5; September-October; 2016; Page No. 04-09

A Comparative Analysis of different Sorting algorithm- A survey

Richa Garg

Jayoti Vidyapeeth Women's, University, Jaipur, Rajasthan, India

Abstract

Sorting refers to the arrangement of data in a particular manner. It can be in ascending or descending order. Sorting is important to optimize the data searching results at high level. There are number of sorting algorithms like bubble sort, selection sort, insertion sort etc. This research paper focus on five sorting algorithms i.e. Bubble sort, Selection sort, Insertion sort, Merge sort, Quick sort. And also give the performance analysis of these algorithms with respect to time complexity. Sorting algorithms are always an area of focus for a long time of which one is good to use and when.

KEYWORDS: Algorithm, Sorting, Bubble sort, Insertion sort, Selection sort, Merge sort, Quick sort

Introduction

The name algorithm was given after the name of Abu Ja'far Muhammad ibn Musa Al-Khwarizmi in 9th century.An algorithm consists of sequence of computational steps that converts input into the output. An algorithm is unambiguous and must complete in finite number of instructions.

An algorithm is efficient and fast, if it takes less time to execute and consumes less memory space.

The performance of an algorithm is measured on the basis of time complexity which is a way to represent the amount of time needed to complete the program and space complexity which is the amount of memory space required by the algorithm, during execution **[1].**Sorting is generally understood to be the process of rearranging a given set of objects in a particular order and therefore, the analysis and design of useful sorting algorithms remained one of the important research areas.

There are so many applications of sorting in our daily life also like objects stored in Telephone directories, income tax files, tables of contents, libraries, dictionaries.**[2]**

Sorting Techniques

1. Bubble Sort

Bubble sort is a sorting technique which sorts the array of n number of elements. Inthis sorting algorithm, first element is compared with second elements in the array. If first element is greater than second then swap or interchange otherwise no swapping will take place and this process continues until no swapping required. It is called bubble sort because the smaller element bubbled to the top of the list.

After (N-1) comparisons, the largest element among all the element will descends to the bottom of array**. [3]**

Algorithm

```
// n is total number of elements in the array
int k=0, m=0;
int Temp;
For (k=0; k<n; k++)
{
For (m=0; m<n-i-1; m++)
{
If (a [m]>a [m+1])
{
   Set   Temp=a [m];
   Set   a [m] =a [m+1];
Set   a [m+1] =temp;
}
}
}
```

Example

Take an array of numbers 13,5,9,2,10 and sort the array in ascending order.Elements which are written in bold are being compared**[4]**

First Pass

Corresponding author: Richa Garg

(**135** 9 2 10) ⟶(**513** 9 2 10), Here, algorithm compares the first two elements, and swaps since 13 > 5.
(5 **13 9** 2 10) ⟶(5**9 13** 2 10), Swap since 13 > 9
(5 9**13 2** 10) ⟶(5 9**2 13** 10), Swap since 13 > 2
(5 9 2**13 10**) ⟶(5 9 2 **10 13**), Swap since 13>10
Second Pass
(**5 9** 2 10 13) ⟶(**5 9** 2 10 13), no swapping since 5<9
(5 **9 2** 10 13) ⟶(5**2 9** 10 13), Swap since 9 > 2
(5 2 **9 10** 13) ⟶(5 2 **9 10** 13), no swapping since 9<10
(5 2 9**10 13**) ⟶(5 2 9**10 13**), no swapping since 10<13

Third Pass
(**5 2** 9 10 13) ⟶(**2 5** 9 10 13), swap since 5>2
(2 **5 9** 10 13) ⟶(2 **5 9** 10 13), no Swapping since 5<9
(2 5**9 10** 13) ⟶(2 5**9 10** 13), no swapping since 9<10
(2 5 9 **10 13**) ⟶(2 5 9 **10 13**), no swapping since 10<13
now the array is sorted but algorithm does not know it is sorted or not. So algorithm goes one more pass without any swapping to know it is sorted.
Fourth Pass
(**2 5** 9 10 13)⟶(**25** 9 10 13), no swapping since 2<5
(2 **5 9** 10 13) ⟶(2 **5 9** 10 13), no Swapping since 5<9
(2 5 **9 10** 13) ⟶(2 5 **9 10** 13), no swapping since 9<10
(2 5 9 **10 13**) ⟶(2 5 9 **10 13**), no swapping since 10<13

Analysis

In this sorting algorithm, in first pass (n-1) comparisons will be done, in second pass (n-2) comparisons will be done and so on.
So
(n-1)+ (n-2) + (n-3) +.....+3+2+1
Sum= n (n-1)/2
It means O (n^2)
So complexity of bubble sort is O(n^2)
Best case: when the array is already sorted.
Complexity is O (n)
Average Case: O (n^2)
Worst Case: O (n^2) when the array is in descending order

Advantages:

- Simple and easy to implement

Disadvantages:

- Not suitable for large number of elements
- General complexity is O(n)

2. Selection Sort

This is the simplest algorithm. In this algorithm, first we select the smallest element from the list of array then compare it with first element and swap it or interchange. Again find second smallest element then compare with second element and swap or interchange and continue this procedure until the list is sorted.

Algorithm

```
//n is the total number of elements in the array
int k=0, m=0;
int smallest, temp;
for (k=1; k<n-1; k++)
{    smallest = k;
 For (m=k+1; m<n;m++)
{
If (a[m] <a [smallest])
{
        Smallest = m;
              }
          }
temp= a[k];
a [k] = a [smallest];
a[smallest] =temp;
}
```

Example

Take an array of elements 13, 5, 9,2,10 and sort the array in ascending order. Elements which are written in bold are selected smallest element.

Original array

(13, 5, 9, **2**, 10)

After first pass

(2, **5,** 9, 13, 10) //first smallest element found is 2 so it is replaced by first element 5

After second pass

(2, 5, **9**, 13, 10) // second smallest element found is 5 and it is already at second position so no replacement

After third pass

(2, 5, 9, 13, **10**) // third smallest element found is 9 and it is already at third position so no replacement

After fourth pass

(2, 5, 9, 10, 13) // fourth smallest element found is 10 and it is replaced by fourth element
Now array is sorted in ascending order

Analysis

Selecting the smallest element requires scanning all *n* elements of the array.Then find the next smallest element requires scanning the remaining (n-1) elements and so on.
So T (n) = n+ (*n* − 1) + (*n* − 2) + ... + 2 + 1
T(n)=[n+1]+[(n-1)+2]+[(n-2)+3]+.....
=(n+1)+(n+1)+(n+1)+.....
T(n)=n(n+1)/2 =((n^2) +n)/2 = ((n^2)/2) + (n/2)
So, complexity of selection sort is O(n^2).
Best case: O (n^2)
Average case: O (n^2)
Worst case: O (n^2)
In selection sort, complexity is same in all the three cases.

Advantages:

- It is simple sort
- No additional temporary storage is required

Disadvantages:

- Suitable only for smaller arrays
- General complexity in O(n^2)

1. Insertion Sort

In this sorting algorithm, we sort the array by shifting one by one element.
In this we make a key and then compare all the elements ahead of key. Shift the key to right side.

Algorithm

```
// n is the total number of elements
Int k=0, m=0;
Int key;
for (k=1; k=n-1; k++)
{
      Key=a[k];
 m = k-1;
     while (m>=0 && key<a[m])
      {
           a[m+1]=a[m];
            m--;
           a [m+1] =key;
        }
   }
```

Example

Take an array of elements 13, 5, 9,2,10 and sort the array in ascending order.
Elements which are written in bold are key and compare key with all the elements ahead of it. We start with making second element as key.

13 **5** 9 2 10 //5 is a key compare 5 and 13, 5<13 so swap
513 **9** 2 10 // 9 is a key compare it with 13 and<13 so swap
59 13 **2** 10 // 2 is a key compare it with 13, 9, 5 and 2<13 so swap
59 2 13 10 // 2<9 so swap
5 2 9 13 10 // 2<5 so swap
2 5 9 13 **10 //** 10 is a key compare it with 13,9,5,2 and 10<13 so swap
2 5 9 10 13 //Now the array is sorted.

Analysis

For last element it takes (n-1) comparisons and for second last element it takes (n-2) comparisons and so on
So (n-1) + (n-2)+……+3+2+1= n(n-1)/2
Complexity O(n^2)
Best Case: O(n) when the array is already in ascending order
Average case: O(n^2)
Worst case: O(n^2) when the array is in descending order

Advantages:

- Have good performance while dealing with small elements

Disadvantages:

- Suitable for small number of elements
- General complexity is O(n^2)

2. Merge Sort

Merge sort is a sorting algorithm which is based on divide and conquers technique. It was invented by John Von Neumann in 1945. It happens in three steps:

- First divide the unsorted array into n number of sub array
- Conquer the sub arrays by solving them recursively.
- Merge the sorted sub arrays into sub array until we get only one sub array.

Algorithm

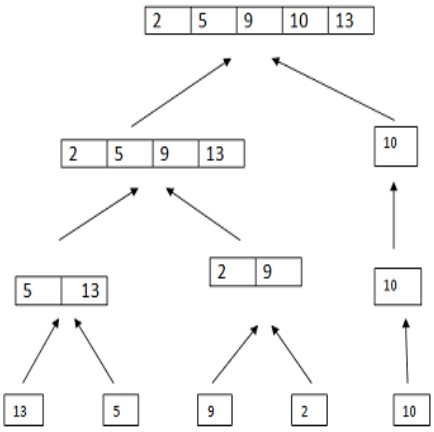
Algorithm of merge sort is given below **[5]**
// a [] is the array, p is starting index, that is 0,

and r is the last index of array//

```
void mergesort (int a[], int p, int r)
{int q;
if (p < r)
 {
   q = floor ((p+r)/2);
mergesort (a, p, q);
mergesort (a, q+1, r);
   merge(a, p, q, r);
 }
}

void merge (int a[], int p, int q, int r)
{
 int b[5];    //same size of a[]
 int i, j, k;
 k = 0;
 i = p;
 j = q+1;
while (i <= q && j <= r)
 {
if (a[i] < a[j])
   {
b [k++] = a [i++];      // same as b[k] =a[i]; k++; i++;
   }
   else
   {
b [k++] = a[j++];
   }
 }

while (i <= q)
 {
   b[k++] = a[i++];
 }

 while(j <= r)
 {
   b[k++] = a[j++];
 }

for (i=r; i >= p; i--)
 {
   a[i] = b[--k];        // copying back the sorted list to
a[]
 }

}
```

Example

Take an array of elements 13, 5, 9,2,10 and sort the array using merge sort in ascending order.
First divide the array into 5 sub lists.



Analysis Merge sort first divide the array in two halves which is divided logn times. Secondly we merge the array. Each element is merged in the sorted list n times so n operations are required.
Best case: O(nlogn)
Average case: O(nlogn)
Worst case: O(nlogn)

Advantages:

- Has O(nlogn) time complexity

Disadvantages:

- Takes more memory space
- Has high space complexity

3. Quick sort

Quick sort also work on divide and conquer algorithm. In this algorithm we choose an element called pivot element, move the smaller elements to the left of the pivot and greater elements to the right of the pivot. Then sort the left and right list recursively.

Algorithm

Quick sort algorithm is given below **[6]**

```
// a[] is the array, p is starting index, that is 0,
and r is the last index of array//

void quicksort (int a[], int p, int r)
{
if (p < r)
 {
   int q;
   q = partition (a, p, r);
```

```
quicksort (a, p, q);
quicksort (a, q+1, r);
 }
}

int partition (int a[], int p, int r)
{
 int i, j, pivot, temp;
 pivot = a[p];
 i = p;
 j = r;
while (1)
 {
while (a[i] < pivot && a[i]! = pivot)
  i++;
while (a[j] > pivot && a[j]!= pivot)
  j--;
if (i < j)
  {
   temp = a[i];
   a[i] = a[j];
   a[j] = temp;
  }
  else
  {
   return j;
  }
 }
}
```

Example

Take an array of elements 13,5,9,2,10 and sort the elements in ascending order.
(13,5,9,2,10)
As first element 13, middle element 9 and last element 10
So median of [13,9, 10] is 10
(5, 2, 9, **10**,13) //Pivot is 10
Now sort the elements before pivot and after pivot
(2, 5, 9), **10**, (13)
Now the array is sorted.

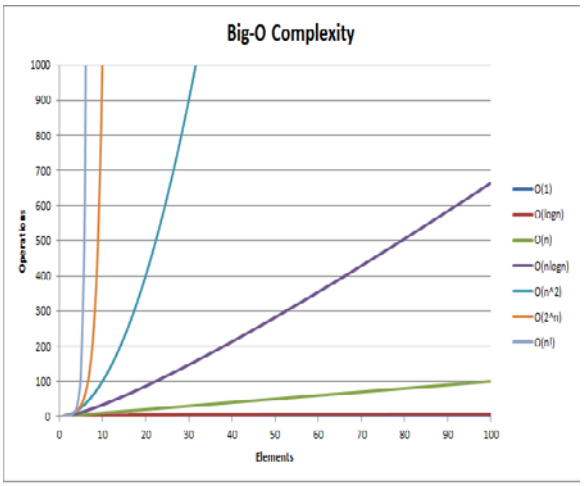
Analysis:

Best case:O(n log n)
when the array is already sorted or when the pivot element is in middle
Average case: O (n logn)
Worst case: O (n^2)
When the pivot element is smallest

Comparison:

	Best case	Average case	Worst case
Bubble sort	O(n)	O(n^2)	O(n^2)
Selection sort	O(n^2)	O(n^2)	O(n^2)
Insertion sort	O(n)	O(n^2)	O(n^2)
Merge sort	O(nlogn)	O(nlogn)	O(nlogn)
Quick sort	O(nlogn)	O(nlogn)	O(n^2)

Graph analysis:



Conclusion:

From the above analysis, we can conclude that bubble sort is the slowest and suitable only for small number of elements. Selection sort and Insertion sort are good for small elements and not good for large elements. Merge sort is complicated and take more space but good for large elements.
Quick sort is the fastest algorithm and good for large number of elements.

References:

1. http://www.studytonight.com/data-structures/introduction-to-data-structures
2. Miss. Pooja K. Chhatwani, Miss. Jayashree S. Somani, "Comparative Analysis & Performance of Different Sorting", *International Journal of Advanced Research in Computer Science and Software Engineering,*Volume 3, Issue 11, November 2013
3. Miss. Pooja K. Chhatwani, Miss. Jayashree S. Somani, "Comparative Analysis & Performance of Different Sorting", *International Journal of Advanced Research in Computer Science and Software Engineering,*Volume 3, Issue 11, November 2013

4. https://en.wikipedia.org/wiki/Bubble_sort
5. http://www.studytonight.com/data-structures/merge-sort
6. http://www.studytonight.com/data-structures/quick-sort