Bilateral Selection Sort

Akshay Zade¹, Vinod Mandloi², Prof. Rupesh Bhoir³
¹, ²Department of Computer Application, Sardar Patel Institute of Technology, Mumbai, India
³Assistant Professor, Sardar Patel Institute of Technology, Mumbai, India

Abstract: In today's digital world data is playing wide role and the amount of the is growing drastically. Sorting of data makes it easier to handle data in large scale. There are many sorting techniques are available for sorting of data. Some of these techniques are Bubble Sort, Insertion Sort, Selection Sort, Quick Sort, Merge Sort, Heap Sort, etc. Some sorting techniques work faster for small amount of data and some work faster for large amount of data.

I. INTRODUCTION
Sorting is process of arranging list of elements to ascending and descending order. Information/data growth in digital world leads to rapid development of sorting algorithm. The sorting algorithms with increased performance and decreased complexity attracts researcher's attention towards sorting techniques. Selection sort algorithm is used to sort elements based on min elements for ascending sort and max element for descending sort. On other hand bilateral sorting implies to sort elements by finding smallest and largest element at same time and swapping with start and end position of unsorted array.

II. ANALYSIS OF OLD SELECTION SORT ALGORITHM [1]

A. Selection Sort
This is a very easy sorting algorithm to understand and is very useful when dealing with small amounts of data. However, as with Bubble sorting, a lot of data really slows it down. Selection sort does have one advantage over other sort techniques. Although it does many comparisons, it does the least amount of data moving. Thus, if your data has small keys but large data area, then selection sorting may be the quickest.

B. Pseudo Code of Old Selection Sort
Algorithm SelectionSort (X, n) =>X [0...n-1]
For i ← n – 1 to 0
IndexOfLarge ← 0
For j←1 to i
If (X[j]>X [IndexOfLarge])
IndexOfLarge ← large ←X [IndexOfLarge]
X [IndexOfLarge] ← X[i]
X[i] ← Large

III. ANALYSIS OF BILATERAL SELECTION SORT (BSS) ALGORITHM
The idea behind the BSS is to make a better version of selection sort and reduce the time complexity of selection sort by minimizing the number of iterations i.e. by selecting smallest and largest element at same time and putting them to their appropriate place to get sorted elements. The time complexity of selection sort is O(N²) whereas the time complexity of BSS is also O(N²). Even if it’s O(N²) the bilateral performs better because there are less number of iterations as two elements are sorted at a time.

A. Algorithm
1) Step 1: Set MIN AND MAX to location 0
2) Step 2: Set leftshrink = i and rightshrink= n
3) Step 3: Search the minimum and maximum element in the list
4) Step 4: Swap the minimum with List[leftshrink] and Maximum with List[rightshrink]
5) Step 5: Increment min point and decrement max point
6) Step 6: Decrement rightshrink
7) Step 7: Repeat until List/2
B. Pseudo code

START PROCEDURE
    list: Array of elements
    n: Size of list
    rightshrink: n-1
    FOR i = 0 till n/2
        min = i
        max = i
        FOR leftshrink = i till rightshrink
            IF list[leftshrink] > max
                THEN max = list[leftshrink] AND getindexmax = leftshrink
            ELSE IF list[leftshrink] < min
                THEN min = list[leftshrink] AND getindexmin = leftshrink
        END FOR
        Swap in list (list[i] with list[getindexmin])
        IF list[getindexmax] == max
            Swap in list (list[rightshrink] with list[getindexmin])
        ELSE
            Swap in list (list[rightshrink] with list[getindexmax])
        ENDIF
        rightshrink = rightshrink - 1
    END FOR
END PROCEDURE

C. Code

```java
public void BSS(int[] arr)
{
    int n = arr.length;
    int rightshrink=n-1;
    int temp=0,min,max,getindexmin,getindexmax;
    for(int i = 0;i < n/2; i++) {
        min = max = arr[i];
        getindexmin = getindexmax = i;
        for (int leftshrink = i; leftshrink <= rightshrink; leftshrink++){
            if (arr[leftshrink] > max){
                max = arr[leftshrink];
                getindexmax = leftshrink;
            }
            else if (arr[leftshrink] < min){
                min = arr[leftshrink];
                getindexmin = leftshrink;
            }
        }
        temp = arr[i];
        arr[i]=arr[getindexmin];
        arr[getindexmin]=temp;
        if (arr[getindexmin] == max) {
            temp = arr[rightshrink];
            arr[rightshrink]=arr[getindexmin];
            arr[getindexmin]=temp;
        }
    }
}
```
void heapify(int arr[], int N, int i)
{
    int largest = i; // Initialize largest as root
    int left = 2 * i + 1; // left = 2*i + 1
    int right = 2 * i + 2; // right = 2*i + 2
    // If left child is larger than root
    if (left < N && arr[left] > arr[largest])
        largest = left;
    // If right child is larger than largest so far
    if (right < N && arr[right] > arr[largest])
        largest = right;
    // If largest is not root
    if (largest != i)
    {
        swap(arr[i], arr[largest]); // swap
        heapify(arr, N, largest); // recursive call
    }
}

int main()
{
    int arr[] = {10, 20, 30, 40, 50};
    int N = sizeof(arr)/sizeof(arr[0]);
    heapify(arr, N, 0);
    printArray(arr, N);
    return 0;
}

else {
    temp = arr[rightshrink];
    arr[rightshrink] =arr[getindexmax];
    arr[getindexmax] =temp;
} --rightshrink;
}

IV. BREAK DOWN ANALYSIS OF SELECTION SORT WITH BILATERAL SELECTION SORT
BSS Sort works with same structure as Selection sort follows. But the iterations are dynamically calculated on the size of data which needs to be sort. Selection Sort has the complexity of O(n^2) for the best case, worst case and average case. BSS complexity is been derived (n/2)*(n/2) = O(n^2).
Following is the breakdown for every iteration for better understanding.

A. Considering Data of 5 Elements

| Selection Sort |
|----------------|
| Outer Loop     | 1   | 2   | 3   | 4   |
| Inner Loop     | 4   | 3   | 2   | 1   |
| Total          | 4   | 7   | 6   | 5   |
| Outer Loop total(4) x (Total Inner Loop(4+3+2+1)) = 14 |

| Bilateral Selection Sort |
|--------------------------|
| Outer Loop               | 1   | 2   |
| Inner Loop               | 5   | 3   |
| Total                    | 5   | 8   |
| Outer Loop total(2) x (Total Inner Loop(5+3)) = 10 |

B. Considering Data of 6 Elements

| Selection Sort |
|----------------|
| Outer Loop     | 1   | 2   | 3   | 4   | 5   |
| Inner Loop     | 5   | 4   | 3   | 2   | 1   |
| Total          | 5   | 9   | 8   | 7   | 6   |
| Outer Loop total(5) x (Total Inner Loop(5+4+3+2+1)) = 20 |

| Bilateral Selection Sort |
|--------------------------|
| Outer Loop               | 1   | 2   | 3   |
| Inner Loop               | 6   | 4   | 2   |
| Total                    | 7   | 12  | 10  |
| Outer Loop total(3) x (Total Inner Loop(6+4+2)) = 15 |
C. Considering Data of 7 Elements

**Selection Sort**

| Outer Loop | 1 | 2 | 3 | 4 | 5 | 6 |
|------------|---|---|---|---|---|---|
| Inner Loop | 6 | 5 | 4 | 3 | 2 | 1 |

Total: Outer Loop total(6) x (Total Inner Loop(6+5+4+3+2+1)) = 27

**Bilateral Selection Sort**

| Outer Loop | 1 | 2 | 3 |
|------------|---|---|---|
| Inner Loop | 7 | 5 | 3 |

Total: Outer Loop total(3) x (Total Inner Loop(7+5+3)) = 18

D. Considering Data of 8 Elements

**Selection Sort**

| Outer Loop | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|------------|---|---|---|---|---|---|---|
| Inner Loop | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

Total: Outer Loop total(7) x (Total Inner Loop(7+6+5+4+3+2+1)) = 35

**Bilateral Selection Sort**

| Outer Loop | 1 | 2 | 3 | 4 |
|------------|---|---|---|---|
| Inner Loop | 8 | 6 | 4 | 2 |

Total: Outer Loop total(4) x (Total Inner Loop(8+6+4+2)) = 24

E. Considering Data of 9 Elements

**Selection Sort**

| Outer Loop | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|------------|---|---|---|---|---|---|---|---|
| Inner Loop | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

Total: Outer Loop total(8) x (Total Inner Loop(8+7+6+5+4+3+2+1)) = 44

**Bilateral Selection Sort**

| Outer Loop | 1 | 2 | 3 | 4 |
|------------|---|---|---|---|
| Inner Loop | 9 | 7 | 5 | 3 |

Total: Outer Loop total(4) x (Total Inner Loop(9+7+5+3)) = 28
F. Considering Data of 10 Elements

Selection Sort

| Outer Loop  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|-------------|---|---|---|---|---|---|---|---|---|
| Inner Loop  | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| Total       | 54 |

Bilateral Selection Sort

| Outer Loop  | 1 | 2 | 3 | 4 | 5 |
|-------------|---|---|---|---|---|
| Inner Loop  | 10 | 8 | 6 | 4 | 2 |
| Total       | 35 |

As per the understanding from the breakdown analysis we find the pattern of total iteration between both the selection sort and bilateral selection sort as follow:
This data shows the iterations of selection sort with respect to bilateral selection sort and number of iterations in bilateral sort is lesser which makes it faster than selection sort.

V. COMPARISON ANALYSIS OF SELECTION SORT AND BILATERAL SELECTION SORT (BSS)

The following table shows the detailed analysis of Bilateral Selection Sort and its performance with respect to Selection Sort Algorithm in nanoseconds.

The above analysis for the comparison between Bilateral Selection Sort and Selection Sort Algorithm is derived from the following constraints:

Dataset of random 50 to 500 numbers are applied on Bilateral Selection Sort and the same dataset is applied on the Selection Sort for the analysis.

The following dataset is applied 10 times for more accurate result because on every execution of algorithm the CPU internally processes different task on each thread in the operating system so output execution in nano time differs every time so to get precise result the random same dataset is applied for 2 times of 1st Iteration of result.

Formula for execution time of algorithm is followed as:

```
long startTime = System.nanoTime();
BilateralSelectionSortAlgorithm(array);
```

OR

```
SelectionSortAlgorithm(array);
```

```
long elapsedTime = System.nanoTime() - startTime;
```

As per the analysis the derived output is been tested

| No. of data | Selection Sort | Bilateral Selection Sort | Total |
|-------------|----------------|--------------------------|-------|
| 50          | 10             | 10                       | 20    |
| 100         | 10             | 10                       | 20    |
| 150         | 10             | 10                       | 20    |
| 200         | 10             | 10                       | 20    |
| 250         | 10             | 10                       | 20    |
| 300         | 10             | 10                       | 20    |
| 350         | 10             | 10                       | 20    |
| 400         | 10             | 10                       | 20    |
| 450         | 10             | 10                       | 20    |
| 500         | 10             | 10                       | 20    |
| Total       |                |                          | 200   |

Therefore the comparison chart of Bilateral Selection Sort and Selection Sort is derived where it results to Bilateral Selection Sort Algorithm works faster compare to Selection Sort Algorithm in sorting data.
VI. CONCLUSION

Logic of Bilateral Selection Sort is based on the Selection sort algorithm. The main difference in Selection sort and Bilateral Selection Sort is that the Selection sort sorts the data from one end i.e. from largest element of array to smallest element or from smallest to largest but the later starts sorting from both ends and finds the largest and smallest data elements of array in single iteration and places those at their appropriate locations then during second iteration it sorts the second largest and second smallest elements from the remaining array data and places those in their appropriate locations in the array. Similarly, it sorts rest of the data elements and puts those in their proper positions. Bilateral Selection Sort sorts the data in half iterations as compared to selection sort technique. The improvement is also of the order.

REFERENCE

[1] Sultanullah Jadoon, Salman Faiz, Prof. Dr. Salim ur Rehman, Prof. Hamid Jan, Design and Analysis of Optimized Selection Sort Algorithm.
[2] Kirti Kaushik, Jyoti Yadav, Kriti Bhatia Design and Analysis of Optimized Selection Sort Algorithm.
[3] Holcers Balazs, Introduction to Sorting Algorithms: A guide to implement sorting algorithms on a step by step basis.
