ANALYSIS OF ALGORITHMS

Project

Abstract

Analyzing the performance of sorting algorithms. Counting the number of comparisons done on the input data for each comparison type algorithm. Comparing the performance of insertion, modified insertion, selection and bubble sort, Comparing the performance of quick, randomized quick, selection and insertion sort, Comparing the performance of merge, randomized quick modified insertion and heap sorts based on the experimental results and theory.

MANOJ PRADEEP KUMAR DEVARAPALLI
50484138
manojpra.devarapa@smail.astate.edu
HIMAJA KOTTURU
50490169
himaja.kotturu@smail.astate.edu
AMAN HASAN SHAIK
50489517
amanhasa.shaik@smail.astate.edu

ANALYSIS OF ALGORITHMS PROJECT

PRADEEP DEVARAPALLI
50484138
manojpra.devarapa@smail.astate.edu

HIMAJA KOTTURU
50490169
himaja.kotturu@smail.astate.edu

AMAN HASAN SHAIK
50489517
amanhasa.shaik@smail.astate.edu

PROGRAM TO ANALYSE THE PERFORMANCE OF SORTING ALGORITHMS:

```
/*
    Project          : Program to analyze the performance of Sorting algorithms
    Team Members : Aman Hasan Shaik
                         Himaja Kotturu
                         Manoj Pradeep Kumar Devarapalli

*/

#include <iostream>
using std :: cout;
using std :: cin;
#include <random> // used for rand()
#include <cstdlib>
#include <cmath>
#include <ctime>
#include <sys/time.h> // used for calculating time

const int MIN = 0;
const int MAX = 1000;

struct timeval start,end;
double diff;
/* Function to print the original and sorted data */

void print(int arr[],int n)
{
```

PRADEEP DEVARAPALLI
50484138
manojpra.devarapa@smail.astate.edu

HIMAJA KOTTURU
50490169
himaja.kotturu@smail.astate.edu

AMAN HASAN SHAIK
50489517
amanhasa.shaik@smail.astate.edu

```
    for(int a=0;a <n;a++)
    {
        cout<<arr[a] << ((a+1) % 10 == 0 ? "\n": "\t");
    }
}

/* Function for Insertion Sort*/
int insertion_sort(int arr[],int &n,int &count1)
{
    for(int i=1;i<n;i++)
    {
        int j=i;
        while(j>0 && arr[j] < arr[j-1])
        {
            int temp= arr[j];
            arr[j]=arr[j-1];
            arr[j-1]=temp;
            j--;
            count1++;// # of times loop iterates
        }
    }
}
/*Function for selection Sort*/

int selection_sort(int arr[],int &n,int &count2)
{
    for (int i=0; i < n-1; i++)
```

PRADEEP DEVARAPALLI
50484138
manojpra.devarapa@smail.astate.edu

HIMAJA KOTTURU
50490169
himaja.kotturu@smail.astate.edu

AMAN HASAN SHAIK
50489517
amanhasa.shaik@smail.astate.edu

```
        {
          int pos_min = i;
          for (int j=i+1; j < n; j++)
                {
                        if (arr[j] < arr[pos_min])
                                pos_min=j;
           count2++;//#of times loop iterates
                }
                if (pos_min != i)
    {
          int temp = arr[i];
          arr[i] = arr[pos_min];
          arr[pos_min] = temp;
    }
                count2++;
        }
}

/*Function for Merge Sort*/
int merge(int a[], int low, int high, int mid,int &count3,int merge_count)
{

   int i, j, k, c[30000];
   i = low;
   k = low;
   j = mid + 1;
   while (i <= mid && j <= high)
```

PRADEEP DEVARAPALLI
50484138
manojpra.devarapa@smail.astate.edu

HIMAJA KOTTURU
50490169
himaja.kotturu@smail.astate.edu

AMAN HASAN SHAIK
50489517
amanhasa.shaik@smail.astate.edu

```
{
   if (a[i] < a[j])
   {
      c[k] = a[i];
      k++;
      i++;
   }
   else
   {
      c[k] = a[j];
      k++;
      j++;
   }
   merge_count++;

}
while (i <= mid)
{
   c[k] = a[i];
   k++;
   i++;
}
while (j <= high)
{
   c[k] = a[j];
   k++;
   j++;
```

PRADEEP DEVARAPALLI
50484138
manojpra.devarapa@smail.astate.edu

HIMAJA KOTTURU
50490169
himaja.kotturu@smail.astate.edu

AMAN HASAN SHAIK
50489517
amanhasa.shaik@smail.astate.edu

```
    }
    for (i = low; i < k; i++)
    {
        a[i] = c[i];
    }
    count3+=merge_count; // #of times loop iterates
}

int merge_sort(int a[], int low, int high,int &count3)
{
    int mid,merge_count=0;
    if (low < high)
    {
        mid=(low+high)/2;
        merge_sort(a,low,mid,count3);
        merge_sort(a,mid+1,high,count3);
        merge(a,low,high,mid,count3,merge_count);
    }
}
            /* ******Quick Sort*******/
/* Function for swaping elements*/
void swap(int& a, int& b)
{
    int temp = a;
    a = b;
    b = temp;
}
```

ANALYSIS OF ALGORITHMS PROJECT

PRADEEP DEVARAPALLI
50484138
manojpra.devarapa@smail.astate.edu

HIMAJA KOTTURU
50490169
himaja.kotturu@smail.astate.edu

AMAN HASAN SHAIK
50489517
amanhasa.shaik@smail.astate.edu

```
int pivot(int arr[], int first, int last,int &count4)
{
   int  p = first;
   int pivotElement = arr[first];


   for(int i = first+1 ; i <= last ; i++)
   {
      /* If you want to sort the list in the other order, change "<=" to ">" */
      if(arr[i] <= pivotElement)
      {
         p++;
         swap(arr[i], arr[p]);
      }
      count4++;
   }
   swap(arr[p], arr[first]);
}
int quick_Sort( int arr[], int first, int last,int &count4)
{
   int pivotElement;
   cout<<"\n\n\nabc";
   if(first < last)
   {
      cout<<"\n\nQuick Sort";
      pivotElement = pivot(arr, first, last,count4);
      quick_Sort(arr, first, pivotElement-1,count4);
      quick_Sort(arr, pivotElement+1, last,count4);
```

PRADEEP DEVARAPALLI
50484138
manojpra.devarapa@smail.astate.edu

HIMAJA KOTTURU
50490169
himaja.kotturu@smail.astate.edu

AMAN HASAN SHAIK
50489517
amanhasa.shaik@smail.astate.edu

```
    }


}

                /**********Heap Sort**********/
void maxHeapify(int array[], int parentIndex, int arraySize,int &count5)
{
   int maxIndex = parentIndex,
      leftChildIndex = 2*parentIndex + 1,
      rightChildIndex = leftChildIndex + 1;  // 2*parentIndex + 2

   if (leftChildIndex < arraySize && array[maxIndex] < array[leftChildIndex])
      maxIndex = leftChildIndex;

   if (rightChildIndex < arraySize && array[maxIndex] < array[rightChildIndex])
      maxIndex = rightChildIndex;

   if (maxIndex != parentIndex)
   {
      count5++;
      int temp = array[maxIndex];
      array[maxIndex] = array[parentIndex];
      array[parentIndex] = temp;
      maxHeapify(array,maxIndex,arraySize,count5);
   }
}
```

ANALYSIS OF ALGORITHMS PROJECT

PRADEEP DEVARAPALLI
50484138
manojpra.devarapa@smail.astate.edu

HIMAJA KOTTURU
50490169
himaja.kotturu@smail.astate.edu

AMAN HASAN SHAIK
50489517
amanhasa.shaik@smail.astate.edu

```
void buildMaxHeap(int array[], int arraySize,int &count5)
{
   for (int node = arraySize / 2 + 1; node >= 0; node--)
      maxHeapify(array, node, arraySize,count5);
}
void heap_sort(int array[], int arraySize,int &count5)
{
   buildMaxHeap(array, arraySize,count5);
   do
   {
      int temp = array[0];
      array[0] = array[arraySize-1];
      array[arraySize-1] = temp;
      arraySize--;
      maxHeapify(array,0,arraySize,count5);
   } while (arraySize > 1);

}




int main()
{
   srand(time(0));

   int choice1,choice2,choice3;
   int count1=0,count2=0,count3=0,count4=0,count5=0;
```

PRADEEP DEVARAPALLI
50484138
manojpra.devarapa@smail.astate.edu

HIMAJA KOTTURU
50490169
himaja.kotturu@smail.astate.edu

AMAN HASAN SHAIK
50489517
amanhasa.shaik@smail.astate.edu

```
int count=0,n;
clock_t start, end;
cout<<"\nSelect the size of the array from the following:";
cout<<"\n\t1.100\n\t2.1000\n\t3.10000\n\t4.30000\n";
cout<<"Select 1 or 2 or 3 or 4:";
cin >> choice2;
switch(choice2)          //Input size Selection
{
   case 1:
         n=100;
         break;
   case 2:
         n=1000;
         break;
   case 3:
         n=10000;
         break;
   case 4:
         n=30000;
         break;
   default:
         cout <<"ooops..! Invalid size";

}

int original_arr[n];
int temp_arr[n];
```

PRADEEP DEVARAPALLI
50484138
manojpra.devarapa@smail.astate.edu

HIMAJA KOTTURU
50490169
himaja.kotturu@smail.astate.edu

AMAN HASAN SHAIK
50489517
amanhasa.shaik@smail.astate.edu

```
int temp;
int merge_count=0;


cout<<"\nSelect the order of the input data from the following";
cout<<"\n\t1.Assending Order\n\t2.Random Order\n\t3.Decending Order\n";
cout<<"Select 1 or 2 or 3 :";
cin >> choice3;


switch(choice3)     //Order Selection
{
   case 1:
        for(int i=0 ; i<n ; i++)
        {
           original_arr[i] =i;
        }
        break;
   case 2:
        for(int i=0 ; i<n ; i++)
        {
           original_arr[i] = ((rand() % (MAX -MIN + 1)) + MIN);
        }
        break;
   case 3:
        for(int i=0; i<n; i++)
        {


           original_arr[i] = i ;
```

PRADEEP DEVARAPALLI
50484138
manojpra.devarapa@smail.astate.edu

HIMAJA KOTTURU
50490169
himaja.kotturu@smail.astate.edu

AMAN HASAN SHAIK
50489517
amanhasa.shaik@smail.astate.edu

```
        }
        for(int i =0;i<n/2;++i)
        {

           temp = original_arr[i];
           original_arr[i]=original_arr[n-i-1];
           original_arr[n-i-1]=temp;

        }

        break;
    default:
        cout <<"\noopss...! Invalid selection";
}

int insertion_arr[n];
int selection_arr[n];
int merge_arr[n];
int quick_arr[n];
int heap_arr[n];

for(int i=0;i<n;i++)
{
   insertion_arr[i] = original_arr[i];
}

for(int i=0;i<n;i++)
```

PRADEEP DEVARAPALLI
50484138
manojpra.devarapa@smail.astate.edu

HIMAJA KOTTURU
50490169
himaja.kotturu@smail.astate.edu

AMAN HASAN SHAIK
50489517
amanhasa.shaik@smail.astate.edu

```
{
   selection_arr[i] = original_arr[i];
}

for(int i=0;i<n;i++)
{
   merge_arr[i] = original_arr[i];
}

for(int i=0;i<n;i++)
{
   quick_arr[i] = original_arr[i];
}
for(int i=0;i<n;i++)
{
   heap_arr[i] = original_arr[i];
}
cout<<"Select one among the folling sortings:";
cout<<"\n\t1.Insertion Sort\n\t2.Selection Sort\n\t3.Merge Sort\n\t4.Quick Sort\n\t5.Heap Sort\n";
cout <<"Select 1 or 2 or 3 or 4 or 5:";
cin>>choice1;
switch(choice1)     //Algorithm Sellection
{
   case 1:
            /*Insertion Sort*/
        cout<<"\nElements before sorting:\n";
        print(original_arr,n);
```

PRADEEP DEVARAPALLI 50484138 manojpra.devarapa@smail.astate.edu
HIMAJA KOTTURU 50490169 himaja.kotturu@smail.astate.edu
AMAN HASAN SHAIK 50489517 amanhasa.shaik@smail.astate.edu

```
        start = clock();
        insertion_sort(insertion_arr,n,count1);
        end = clock();
        diff = (double(end - start) / CLOCKS_PER_SEC);
        cout<<"\nNumber of iterations for Insertion Sort is  "<< count1 << " And Time taken is " <<diff;


        cout<<"\n\nElements after sorting:\n";
        print(insertion_arr,n);
        break;
    case 2:
            /*Selection Sort*/
        cout<<"\nElements before sorting:\n";
        print(original_arr,n);
        start = clock();
        selection_sort(selection_arr,n,count2);
        end = clock();
        diff = (double(end - start) / CLOCKS_PER_SEC);
        cout<<"\nNumber of iterations for Selection Sort is  "<< count2 << " And Time taken is " <<diff;
        cout<<"\n\nElements after sorting:\n";
        print(selection_arr,n);
        break;
    case 3:
            /*Merge Sort*/
        cout<<"\nElements before sorting:\n";
        print(original_arr,n);
        start = clock();
        merge_sort(merge_arr,count,n,count3);
```

PRADEEP DEVARAPALLI
50484138
manojpra.devarapa@smail.astate.edu

HIMAJA KOTTURU
50490169
himaja.kotturu@smail.astate.edu

AMAN HASAN SHAIK
50489517
amanhasa.shaik@smail.astate.edu

```
        end = clock();
        diff = (double(end - start) / CLOCKS_PER_SEC);
        cout<<"\nNumber of iterations for Merge Sort is  "<< count3 << " And Time taken is " <<diff;
        cout<<"\n\nElements after sorting:\n";
        print(merge_arr,n);
        break;
    case 4:
            /*Quick Sort*/
        cout<<"\nElements before sorting:\n";
        print(original_arr,n);
        start = clock();
        quick_Sort(quick_arr,count,n,count4);
        end = clock();
        diff = (double(end - start) / CLOCKS_PER_SEC);
        cout<<"\nNumber of iterations for Quick Sort is  "<< count4 << " And Time taken is " <<diff;
        cout<<"\n\nElements after sorting:\n";
        print(quick_arr,n);
        break;
    case 5:
            /*Heap Sort*/
        cout<<"\nElements before sorting:\n";
        print(original_arr,n);
        start = clock();
        heap_sort(heap_arr,n,count5);
        end = clock();
        diff = (double(end - start) / CLOCKS_PER_SEC);
        cout<<"\nNumber of iterations for Heap Sort is  "<< count5 << " And Time taken is " <<diff;
```

PRADEEP DEVARAPALLI
50484138
manojpra.devarapa@smail.astate.edu

HIMAJA KOTTURU
50490169
himaja.kotturu@smail.astate.edu

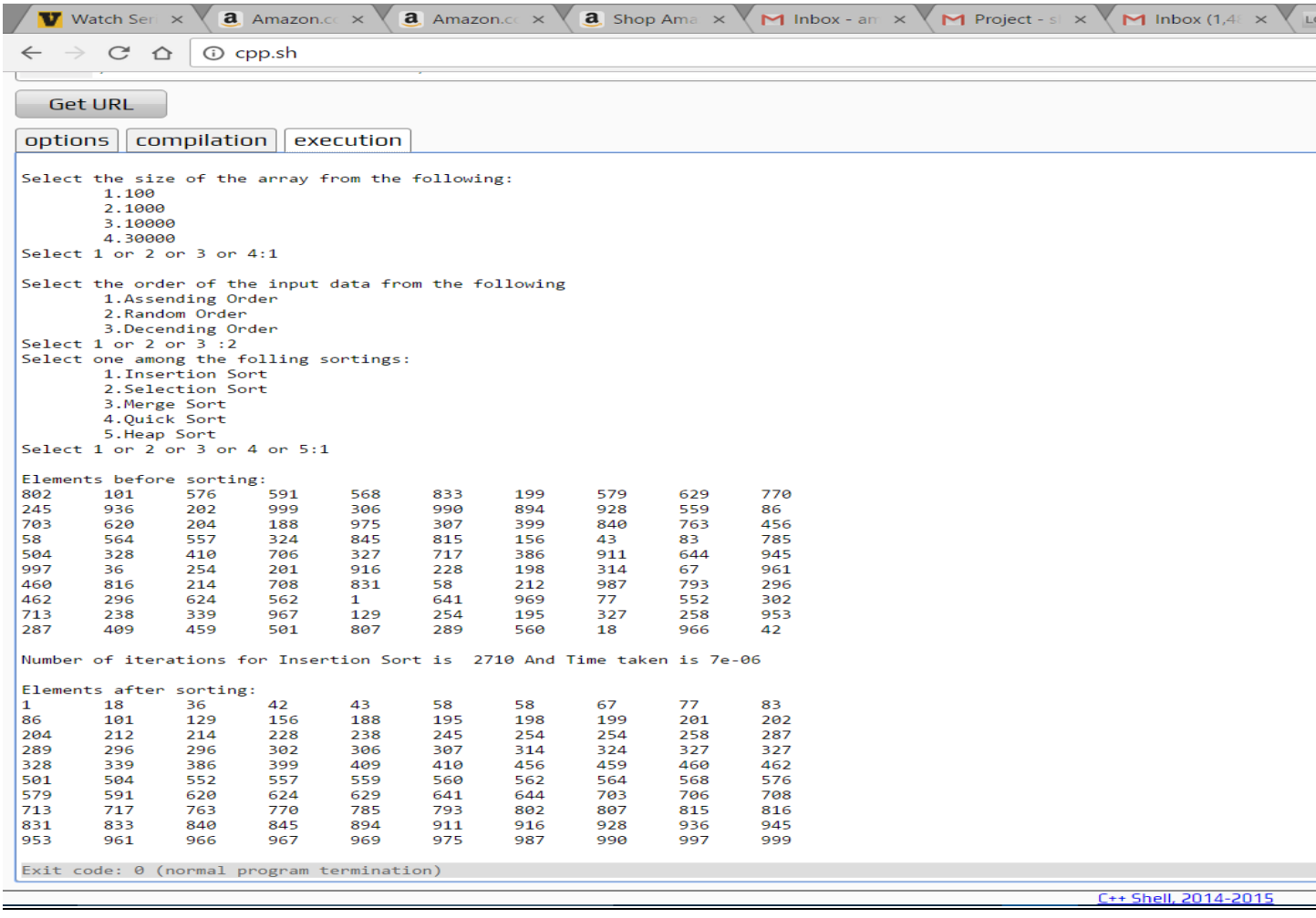
AMAN HASAN SHAIK
50489517
amanhasa.shaik@smail.astate.edu

```
            cout<<"\n\nElements after sorting:\n";
            print(heap_arr,n);
            break;

      default :
            cout<<"oops...! Invalid selection";
   }

}
```

OUTPUT FOR INSERTION SORT:



```
Select the size of the array from the following:
        1.100
        2.1000
        3.10000
        4.30000
Select 1 or 2 or 3 or 4:1

Select the order of the input data from the following
        1.Assending Order
        2.Random Order
        3.Decending Order
Select 1 or 2 or 3 :2
Select one among the folling sortings:
        1.Insertion Sort
        2.Selection Sort
        3.Merge Sort
        4.Quick Sort
        5.Heap Sort
Select 1 or 2 or 3 or 4 or 5:1

Elements before sorting:
802     101     576     591     568     833     199     579     629     770
245     936     202     999     306     990     894     928     559     86
703     620     204     188     975     307     399     840     763     456
58      564     557     324     845     815     156     43      83      785
504     328     410     706     327     717     386     911     644     945
997     36      254     201     916     228     198     314     67      961
460     816     214     708     831     58      212     987     793     296
462     296     624     562     1       641     969     77      552     302
713     238     339     967     129     254     195     327     258     953
287     409     459     501     807     289     560     18      966     42

Number of iterations for Insertion Sort is  2710 And Time taken is 7e-06

Elements after sorting:
1       18      36      42      43      58      58      67      77      83
86      101     129     156     188     195     198     199     201     202
204     212     214     228     238     245     254     254     258     287
289     296     296     302     306     307     314     324     327     327
328     339     386     399     409     410     456     459     460     462
501     504     552     557     559     560     562     564     568     576
579     591     620     624     629     641     644     703     706     708
713     717     763     770     785     793     802     807     815     816
831     833     840     845     894     911     916     928     936     945
953     961     966     967     969     975     987     990     997     999

Exit code: 0 (normal program termination)
```

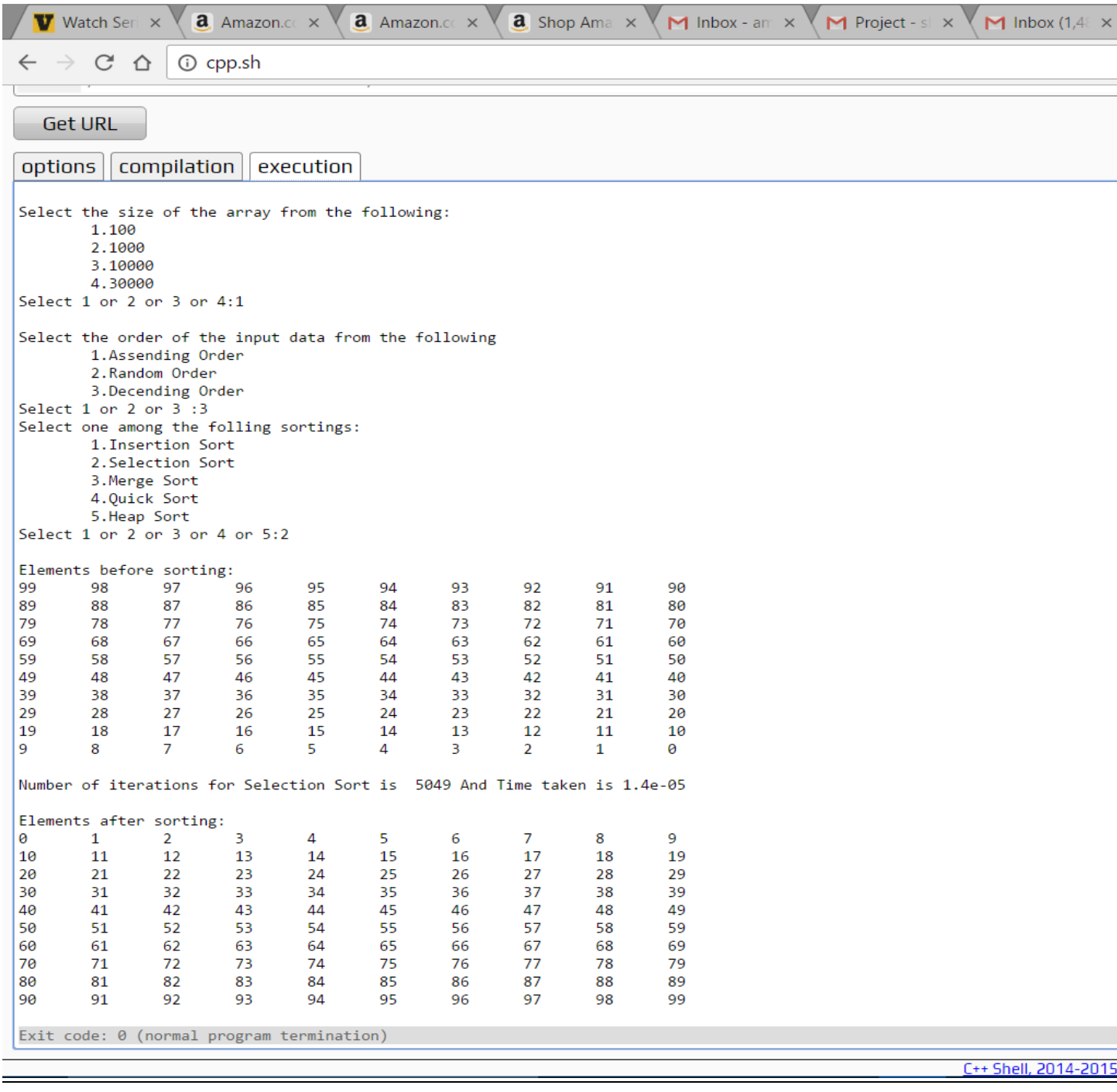
ANALYSIS OF ALGORITHMS PROJECT

PRADEEP DEVARAPALLI
50484138
manojpra.devarapa@smail.astate.edu

HIMAJA KOTTURU
50490169
himaja.kotturu@smail.astate.edu

AMAN HASAN SHAIK
50489517
amanhasa.shaik@smail.astate.edu

OUTPUT FOR SELECTION SORT:



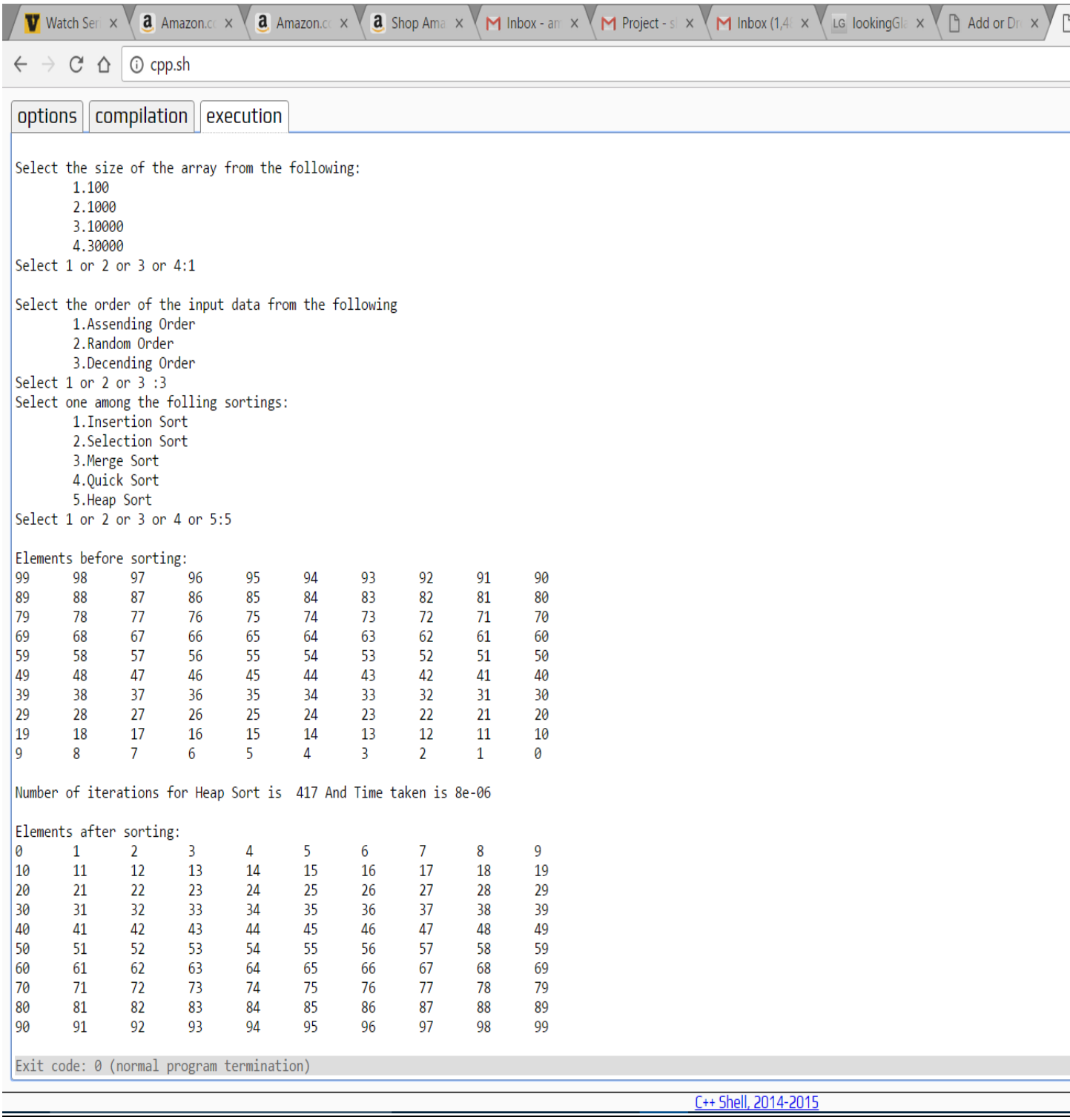
ANALYSIS OF ALGORITHMS PROJECT

PRADEEP DEVARAPALLI
50484138
manojpra.devarapa@smail.astate.edu

HIMAJA KOTTURU
50490169
himaja.kotturu@smail.astate.edu

AMAN HASAN SHAIK
50489517
amanhasa.shaik@smail.astate.edu

OUTPUT FOR HEAP SORT:



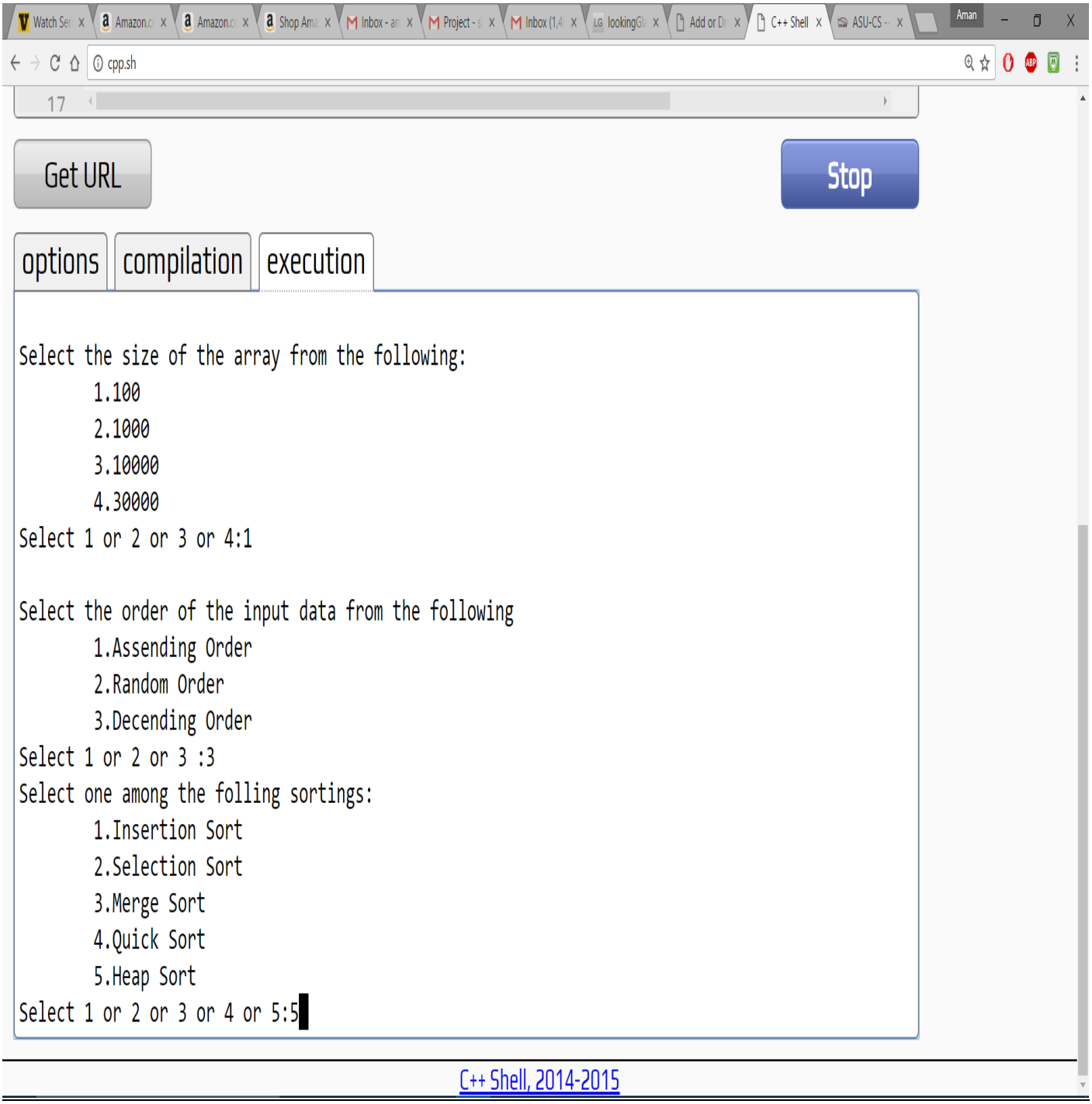
ANALYSIS OF ALGORITHMS PROJECT

PRADEEP DEVARAPALLI
50484138
manojpra.devarapa@smail.astate.edu

HIMAJA KOTTURU
50490169
himaja.kotturu@smail.astate.edu

AMAN HASAN SHAIK
50489517
amanhasa.shaik@smail.astate.edu

USER INTERACTIONS:



ANALYSIS OF ALGORITHMS PROJECT

PRADEEP DEVARAPALLI
50484138
manojpra.devarapa@smail.astate.edu

HIMAJA KOTTURU
50490169
himaja.kotturu@smail.astate.edu

AMAN HASAN SHAIK
50489517
amanhasa.shaik@smail.astate.edu

TOTAL NO OF COMPARISIONS:

Array Size	100			1000			10000			30000		
Array Type	Inc	Dec	Ran	Inc	Dec	Ran	Inc	Dec	Ran	Inc	Dec	Ran
Selection	5049	7390	5389	50499	659216	505781	50004999	52507486	50068028	4500 14999	4575 23364	4502 08983
Bubble	100	14844	11033	1000	1499025	1203 432	10000	1498 45118	1239 16598	30000	1348 814204	1115 842951
Insertion	99	9987	4985	999	999049	511863	9999	9990 0235	5030 3195	29999	8990 98407	4497 15901
Quick	103	102	103	1002	1002	1003	10002	10002	10003	30002	30002	30003
Merge	250	250	250	2500	2500	2500	25000	25000	25000	75000	75000	75000
Heap	653	1364	712	3843	3333	5376			54450			

ANALYSIS OF ALGORITHMS PROJECT

PRADEEP DEVARAPALLI
50484138
manojpra.devarapa@smail.astate.edu

HIMAJA KOTTURU
50490169
himaja.kotturu@smail.astate.edu

AMAN HASAN SHAIK
50489517
amanhasa.shaik@smail.astate.edu

ALGORITHM RUNNING TIME:

Here are the time complexities given for array data structure

Array Size	100			1000			10000			30000		
Array Type	Inc	Dec	Ran	Inc	Dec	Ran	Inc	Dec	Ran	Inc	Dec	Ran
Selection	2.406e-05	3.286e-05	3.162e-05	0.0022	0.0039	0.0023	0.222	0.265	0.224	1.994	2.119	1.996
Bubble	7.53e-07	8.8376e-05	0.00014	5.601e-06	0.0081	0.0086	5.3075e-05	0.815	0.853	0.0001	7.22	7.67
Insertion	1.117e-06	3.052e-05	1.6898e-05	9.666e-06	0.003268	0.0015	9.383e-05	0.302	0.152	0.0003	2.689	1.368
Quick	4.207e-206	4.064e-06	1.0095e-05	5.6159e-05	5.8711e-05	0.0001	0.0007	0.0007	0.0014	0.003	0.003	0.0046
Merge	1.483e-05	1.5635e-05	2.3858e-05	0.0002	0.00019	0.0002	0.0026	0.0027	0.0038	0.0088	0.0088	0.0131
Heap	7.042e-06	1.2974e-05	8.096e-06	4.1254e-05	5.2556e-05	5.7564e-05			0.0005			

Algorithm	Best	Worst	Avg	Method	Remarks

ANALYSIS OF ALGORITHMS PROJECT

PRADEEP DEVARAPALLI
50484138
manojpra.devarapa@smail.astate.edu

HIMAJA KOTTURU
50490169
himaja.kotturu@smail.astate.edu

AMAN HASAN SHAIK
50489517
amanhasa.shaik@smail.astate.edu

Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	Selection	1. Improves the performance of bubble sort and also slow. 2. Unstable but can be implemented as a stable sort. 3. Quite slow for large amount of data.
Bubble	O(n)	$O(n^2)$	$O(n^2)$	Exchange	1. Straightforward, simple and slow. 2. Stable. 3. Inefficient on large tables.
Insertion	O(n)	$O(n^2)$	$O(n^2)$	Insertion	1. Efficient for small list and mostly sorted list. 2. Sort big array slowly. 3. Save memory
Quick	O(n log(n))	O(n log(n))	$O(n^2)$	Partition	1. Fastest sorting algorithm in practice but sometime Unbalanced partition can lead to very slow sort. 2. Available in many slandered libraries. 3. O (log n) space usage. 4. Unstable sort and complex for choosing a good pivot element.

ANALYSIS OF ALGORITHMS PROJECT

PRADEEP DEVARAPALLI
50484138
manojpra.devarapa@smail.astate.edu

HIMAJA KOTTURU
50490169
himaja.kotturu@smail.astate.edu

AMAN HASAN SHAIK
50489517
amanhasa.shaik@smail.astate.edu

Merge	O(n log(n))	O(n log(n))	O(n log(n))	Merge	1. Well for very large list, stable sort. 2. A fast recursive sorting. 3. Both useful for internal and external sorting. 4. It requires an auxiliary array that is as large as the original array to be sorted
Heap	O(n log(n))	O(n log(n))	O(n log(n))	Selection	1. More efficient version of selection sort. 2. No need extra buffer. 3. Its does not require recursion. 4. Slower than Quick and Merge sorts.

ANALYSIS OF ALGORITHMS PROJECT

PRADEEP DEVARAPALLI
50484138
manojpra.devarapa@smail.astate.edu

HIMAJA KOTTURU
50490169
himaja.kotturu@smail.astate.edu

AMAN HASAN SHAIK
50489517
amanhasa.shaik@smail.astate.edu

COMPARISION BETWEEN INSERTION, SELECTION AND BUBBLE:

- Bubble sort is much worse as the number of elements increases, even though both sorting methods have the same asymptotic complexity.
- This analysis is based on the assumption that the input is random - which might not be true all the time.
- When using selecting sort it swaps n times at most. But when using bubble sort, it swaps almost n*(n-1). And obviously reading time is less than writing time even in memory.
- Code size. Bubble-sort is known for its small code footprint.
- Insertion sort is used for smaller data set.

COMPARISION BETWEEN INSERTION, SELECTION AND QUICK:

- Of all the sorting Algorithms Quick sort is the best Algorithm with O(n*log(n)) algorithm on an average case and an O(n2) algorithm in the worst case scenario.
- If a bad pivot is chosen, you can imagine that the “less” subset is always empty. That means we are only creating a subset of one item smaller each time, which gives us O(N^2) behavior in the worst case.
- The quicksort algorithm is complicated, and you have to pass left and right boundary variables, while in other algorithms only need data set and its size.
- Insertion sort is used for smaller data set.
- When using selecting sort it swaps n times at most. But when using bubble sort, it swaps almost n*(n-1). And obviously reading time is less than writing time even in memory.
- One advantage of selection sort against insertion sort, is that the number of writes (swaps) is in O(n), while in insertion sort it is in O(n^2)

COMPARISION BETWEEN MERGE AND HEAP:

- Merge Sort algorithm usually takes the same amount of time independent of the order of the numbers. But with Heap Sort, if the numbers are reversed, the internal heap that it uses has to be re-balanced with each insertion, which adds time to the algorithm.
- Both sort methods have the same time complexity, and are optimal. The time required to merge in a merge sort is counterbalanced by the time required to build the heap in heapsort.
- The merge sort requires additional space. The heapsort may be implemented using additional space, but does not require it. Heapsort, however, is unstable, in that it doesn't guarantee to leave 'equal' elements unchanged.

ANALYSIS OF ALGORITHMS PROJECT

PRADEEP DEVARAPALLI
50484138
manojpra.devarapa@smail.astate.edu

HIMAJA KOTTURU
50490169
himaja.kotturu@smail.astate.edu

AMAN HASAN SHAIK
50489517
amanhasa.shaik@smail.astate.edu

CONCLUSION:

There are two major factors when measuring the performance of a sorting algorithm. The algorithms have to compare the magnitude of different elements and they have to move the different elements around. So counting the number of comparisons and the number of exchanges or moves made by an algorithm offer useful performance measures.

When sorting large record structures, the number of exchanges made may be the principal performance criterion, since exchanging two records will involve a lot of work. When sorting a simple array of integers, then the number of comparisons will be more important.

Sorting algorithms work well on specific type of problem. Some sorting algorithms maintain the relative order of record with equal keys like bubble sort, merge sort, insertion sort. Quick sort, heap sort, selection sort, sorted the input list so that the relative order are not maintain with equal keys.

The selection sort is a good one to use. It is intuitive and very simple to program. It offers quite good performance, its particular strength being the small number of exchanges needed. For a given number of data items, the selection sort always goes through a set number of comparisons and exchanges, so its performance is predictable.

The common sorting algorithms can be divided into two classes by the complexity of their algorithms. The two classes of sorting algorithms are O(n2), which includes the bubble, insertion, selection; and O(n log n) which includes the heap, merge, and quick sorts. Quick sort is efficient only on the average, and its worst case is n2, while heap sort has an interesting property that the worst case is not much different from an average case.