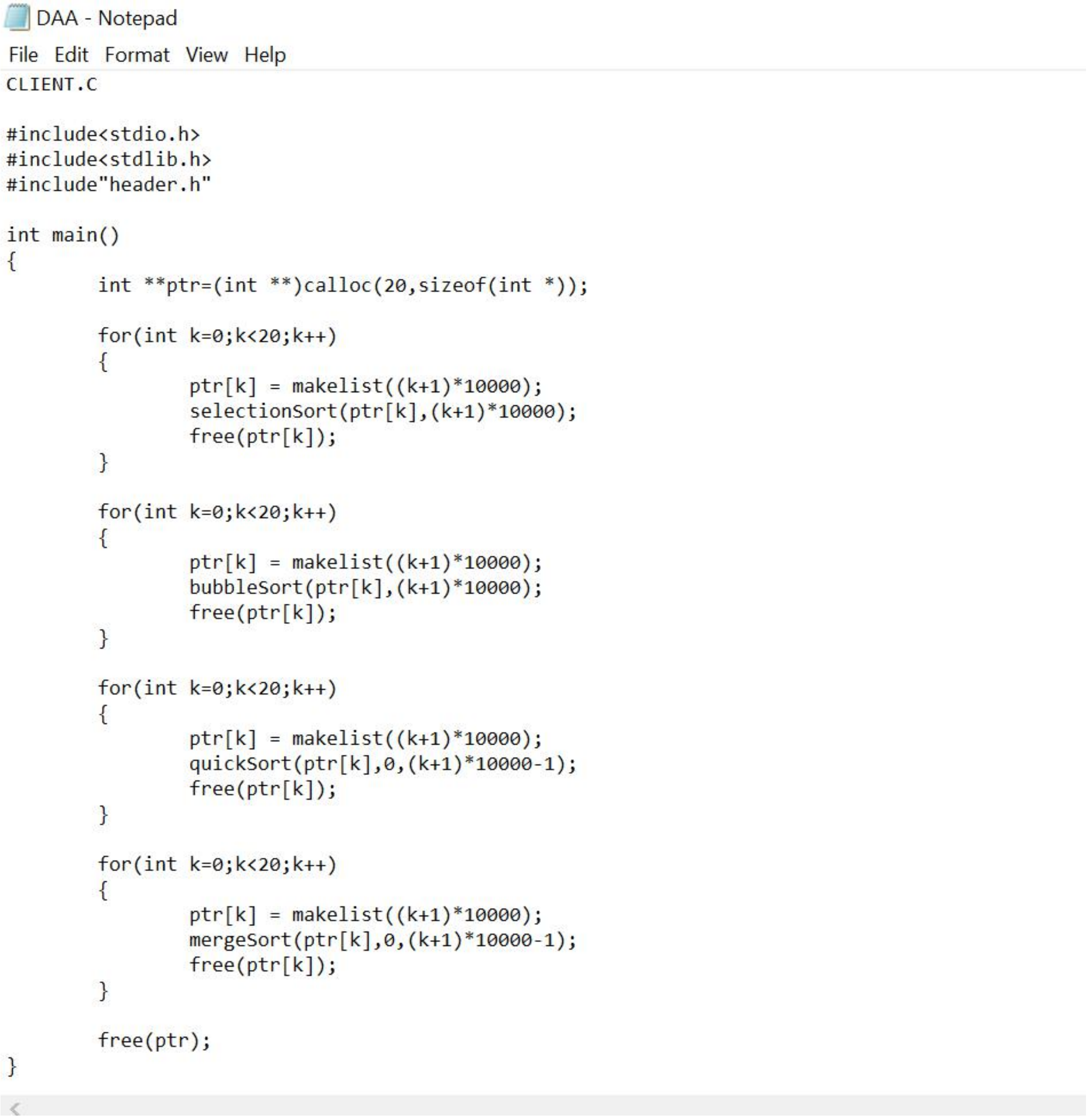
DAA ASSIGNMENT

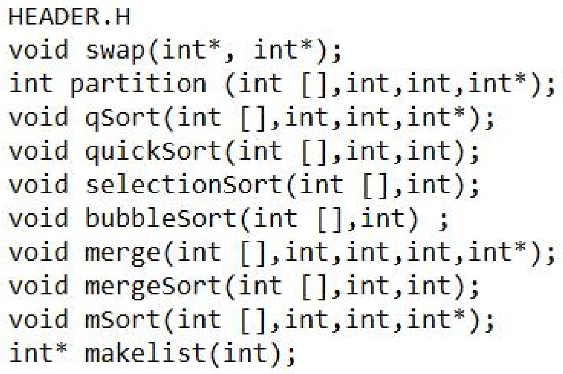
Name : Suhan.B.Revankar
SRN : PES2UG19CS412
Section : G
Date : 25-02-2021

client.c (for all)



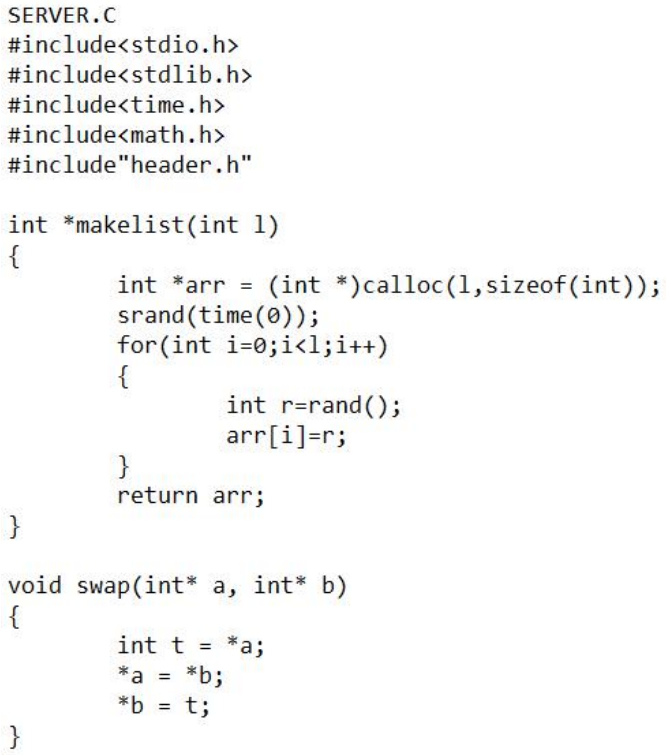
```
CLIENT.C

#include<stdio.h>
#include<stdlib.h>
#include"header.h"

int main()
{
        int **ptr=(int **)calloc(20,sizeof(int *));

        for(int k=0;k<20;k++)
        {
                ptr[k] = makelist((k+1)*10000);
                selectionSort(ptr[k],(k+1)*10000);
                free(ptr[k]);
        }

        for(int k=0;k<20;k++)
        {
                ptr[k] = makelist((k+1)*10000);
                bubbleSort(ptr[k],(k+1)*10000);
                free(ptr[k]);
        }

        for(int k=0;k<20;k++)
        {
                ptr[k] = makelist((k+1)*10000);
                quickSort(ptr[k],0,(k+1)*10000-1);
                free(ptr[k]);
        }

        for(int k=0;k<20;k++)
        {
                ptr[k] = makelist((k+1)*10000);
                mergeSort(ptr[k],0,(k+1)*10000-1);
                free(ptr[k]);
        }

        free(ptr);
}
```

header.c (for all)



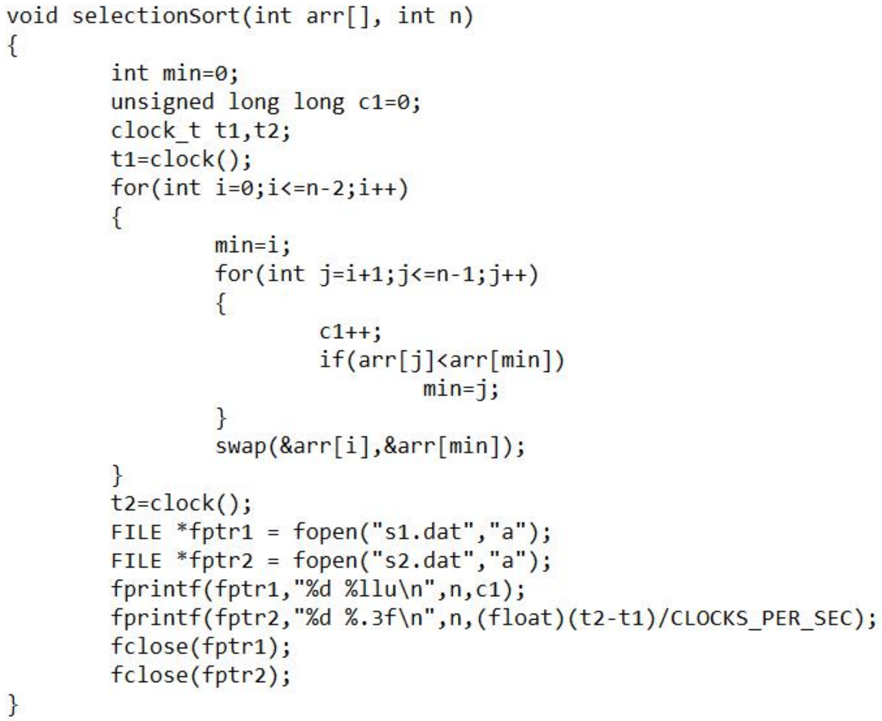
```
HEADER.H
void swap(int*, int*);
int partition (int [],int,int,int*);
void qSort(int [],int,int,int*);
void quickSort(int [],int,int);
void selectionSort(int [],int);
void bubbleSort(int [],int) ;
void merge(int [],int,int,int,int*);
void mergeSort(int [],int,int);
void mSort(int [],int,int,int*);
int* makelist(int);
```

server.c (for list generation)



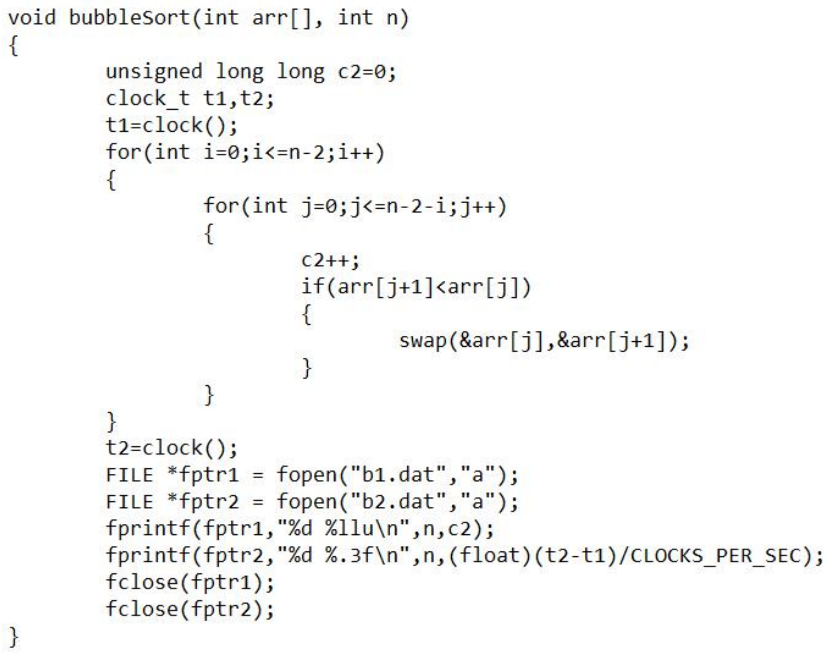
```
SERVER.C
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<math.h>
#include"header.h"

int *makelist(int l)
{
        int *arr = (int *)calloc(l,sizeof(int));
        srand(time(0));
        for(int i=0;i<l;i++)
        {
                int r=rand();
                arr[i]=r;
        }
        return arr;
}

void swap(int* a, int* b)
{
        int t = *a;
        *a = *b;
        *b = t;
}
```

server.c (selection sort)



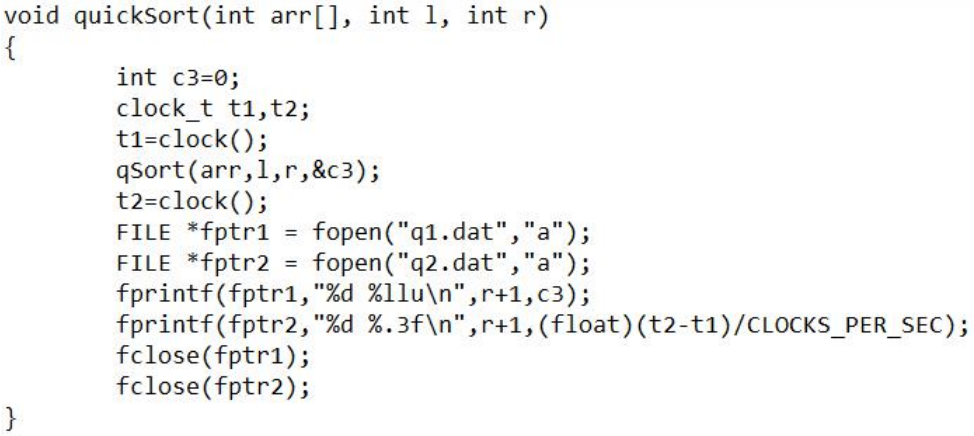
```c
void selectionSort(int arr[], int n)
{
        int min=0;
        unsigned long long c1=0;
        clock_t t1,t2;
        t1=clock();
        for(int i=0;i<=n-2;i++)
        {
                min=i;
                for(int j=i+1;j<=n-1;j++)
                {
                        c1++;
                        if(arr[j]<arr[min])
                                min=j;
                }
                swap(&arr[i],&arr[min]);
        }
        t2=clock();
        FILE *fptr1 = fopen("s1.dat","a");
        FILE *fptr2 = fopen("s2.dat","a");
        fprintf(fptr1,"%d %llu\n",n,c1);
        fprintf(fptr2,"%d %.3f\n",n,(float)(t2-t1)/CLOCKS_PER_SEC);
        fclose(fptr1);
        fclose(fptr2);
}
```

server.c (bubble sort)

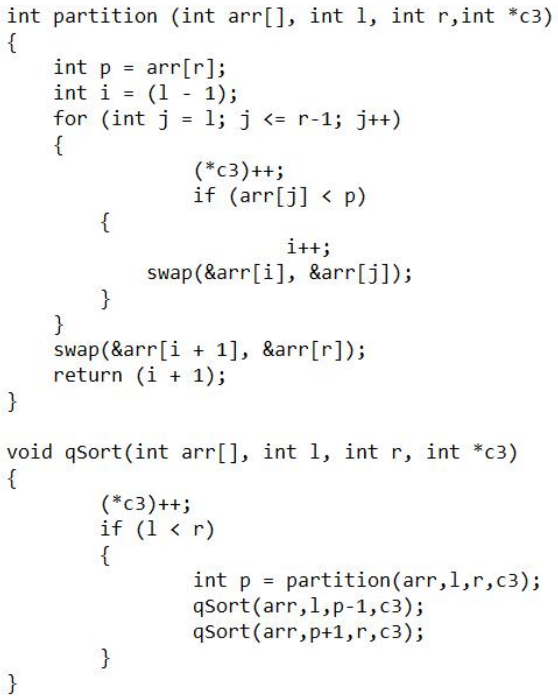


```c
void bubbleSort(int arr[], int n)
{
        unsigned long long c2=0;
        clock_t t1,t2;
        t1=clock();
        for(int i=0;i<=n-2;i++)
        {
                for(int j=0;j<=n-2-i;j++)
                {
                        c2++;
                        if(arr[j+1]<arr[j])
                        {
                                swap(&arr[j],&arr[j+1]);
                        }
                }
        }
        t2=clock();
        FILE *fptr1 = fopen("b1.dat","a");
        FILE *fptr2 = fopen("b2.dat","a");
        fprintf(fptr1,"%d %llu\n",n,c2);
        fprintf(fptr2,"%d %.3f\n",n,(float)(t2-t1)/CLOCKS_PER_SEC);
        fclose(fptr1);
        fclose(fptr2);
}
```

server.c (quick sort)

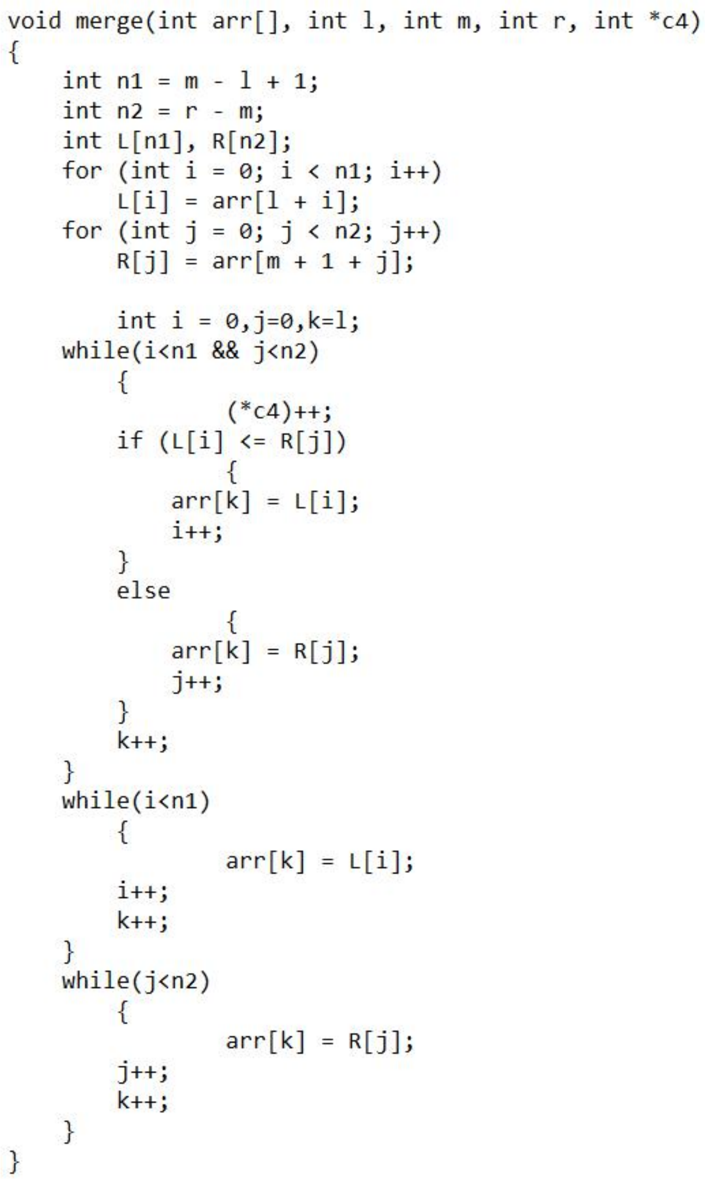


```
void quickSort(int arr[], int l, int r)
{
        int c3=0;
        clock_t t1,t2;
        t1=clock();
        qSort(arr,l,r,&c3);
        t2=clock();
        FILE *fptr1 = fopen("q1.dat","a");
        FILE *fptr2 = fopen("q2.dat","a");
        fprintf(fptr1,"%d %llu\n",r+1,c3);
        fprintf(fptr2,"%d %.3f\n",r+1,(float)(t2-t1)/CLOCKS_PER_SEC);
        fclose(fptr1);
        fclose(fptr2);
}
```



```
int partition (int arr[], int l, int r,int *c3)
{
    int p = arr[r];
    int i = (l - 1);
    for (int j = l; j <= r-1; j++)
    {
                (*c3)++;
                if (arr[j] < p)
        {
                        i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[r]);
    return (i + 1);
}

void qSort(int arr[], int l, int r, int *c3)
{
        (*c3)++;
        if (l < r)
        {
                int p = partition(arr,l,r,c3);
                qSort(arr,l,p-1,c3);
                qSort(arr,p+1,r,c3);
        }
}
```

server.c (merge sort)

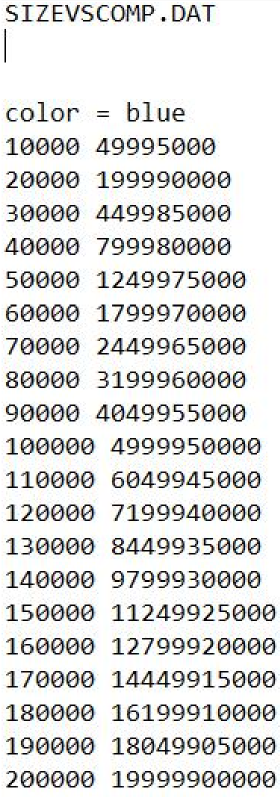


```
void merge(int arr[], int l, int m, int r, int *c4)
{
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];
    for (int i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

        int i = 0,j=0,k=l;
    while(i<n1 && j<n2)
        {
                (*c4)++;
        if (L[i] <= R[j])
                {
            arr[k] = L[i];
            i++;
        }
        else
                {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while(i<n1)
        {
                arr[k] = L[i];
        i++;
        k++;
    }
    while(j<n2)
        {
                arr[k] = R[j];
        j++;
        k++;
    }
}
```

```
void mSort(int arr[], int l, int r, int *c4)
{
    (*c4)++;
        if (l < r)
        {
                int m = l+(r-l)/2;
        mSort(arr,l,m,c4);
        mSort(arr,m+1,r,c4);
        merge(arr,l,m,r,c4);
    }
}

void mergeSort(int arr[], int l, int r)
{
        int c4=0;
        clock_t t1,t2;
        t1=clock();
        mSort(arr,l,r,&c4);
        t2=clock();
        FILE *fptr1 = fopen("m1.dat","a");
        FILE *fptr2 = fopen("m2.dat","a");
        fprintf(fptr1,"%d %llu\n",r+1,c4);
        fprintf(fptr2,"%d %.3f\n",r+1,(float)(t2-t1)/CLOCKS_PER_SEC);
        fclose(fptr1);
        fclose(fptr2);
}
```

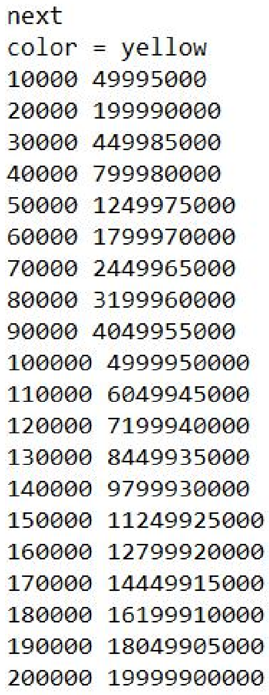
Size of list v/s Number of comparisons

Selection sort



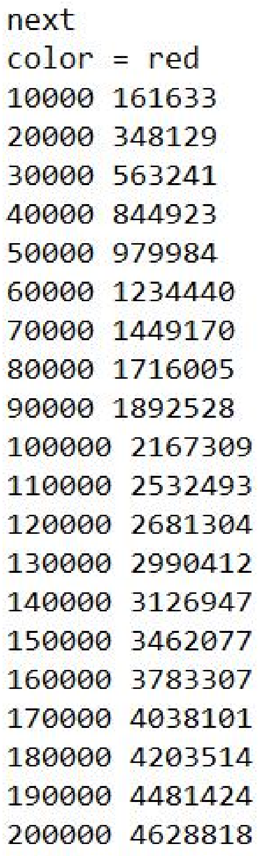
```
SIZEVSCOMP.DAT


color = blue
10000 49995000
20000 199990000
30000 449985000
40000 799980000
50000 1249975000
60000 1799970000
70000 2449965000
80000 3199960000
90000 4049955000
100000 4999950000
110000 6049945000
120000 7199940000
130000 8449935000
140000 9799930000
150000 11249925000
160000 12799920000
170000 14449915000
180000 16199910000
190000 18049905000
200000 19999900000
```

Bubble sort



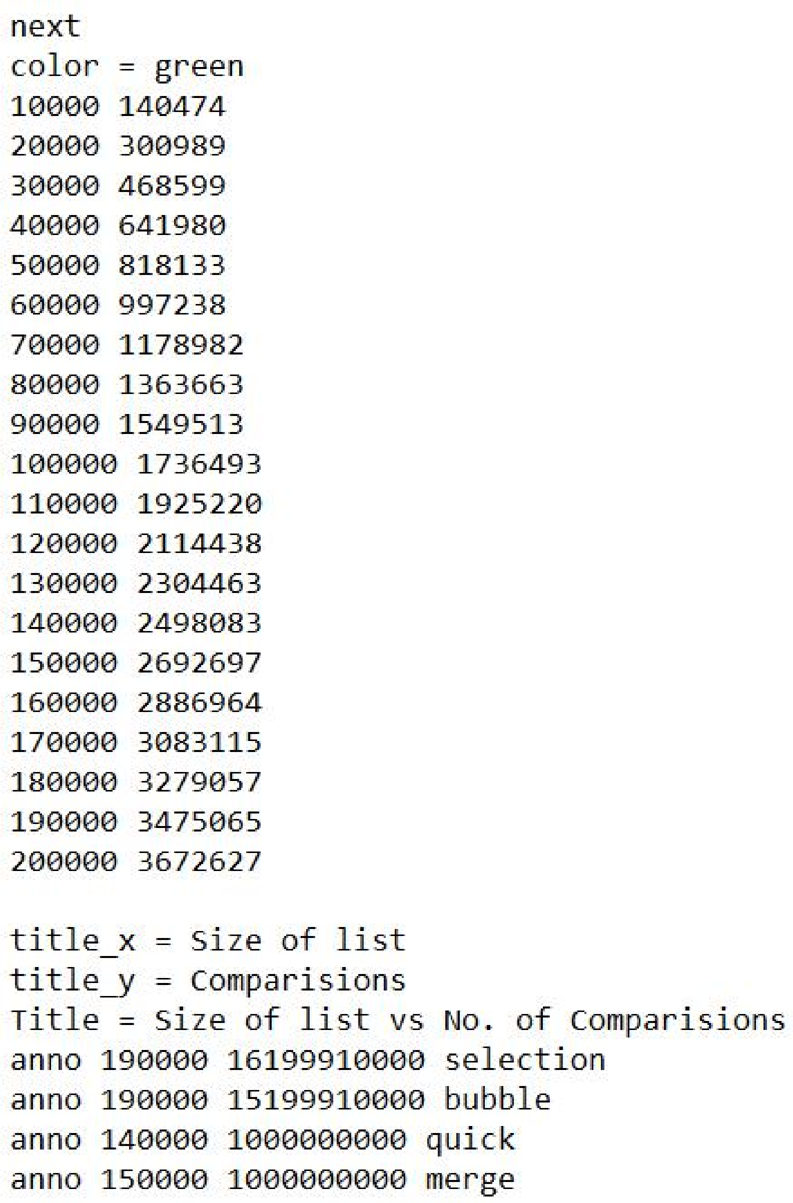
```
next
color = yellow
10000 49995000
20000 199990000
30000 449985000
40000 799980000
50000 1249975000
60000 1799970000
70000 2449965000
80000 3199960000
90000 4049955000
100000 4999950000
110000 6049945000
120000 7199940000
130000 8449935000
140000 9799930000
150000 11249925000
160000 12799920000
170000 14449915000
180000 16199910000
190000 18049905000
200000 19999900000
```

Quick sort



```
next
color = red
10000 161633
20000 348129
30000 563241
40000 844923
50000 979984
60000 1234440
70000 1449170
80000 1716005
90000 1892528
100000 2167309
110000 2532493
120000 2681304
130000 2990412
140000 3126947
150000 3462077
160000 3783307
170000 4038101
180000 4203514
190000 4481424
200000 4628818
```

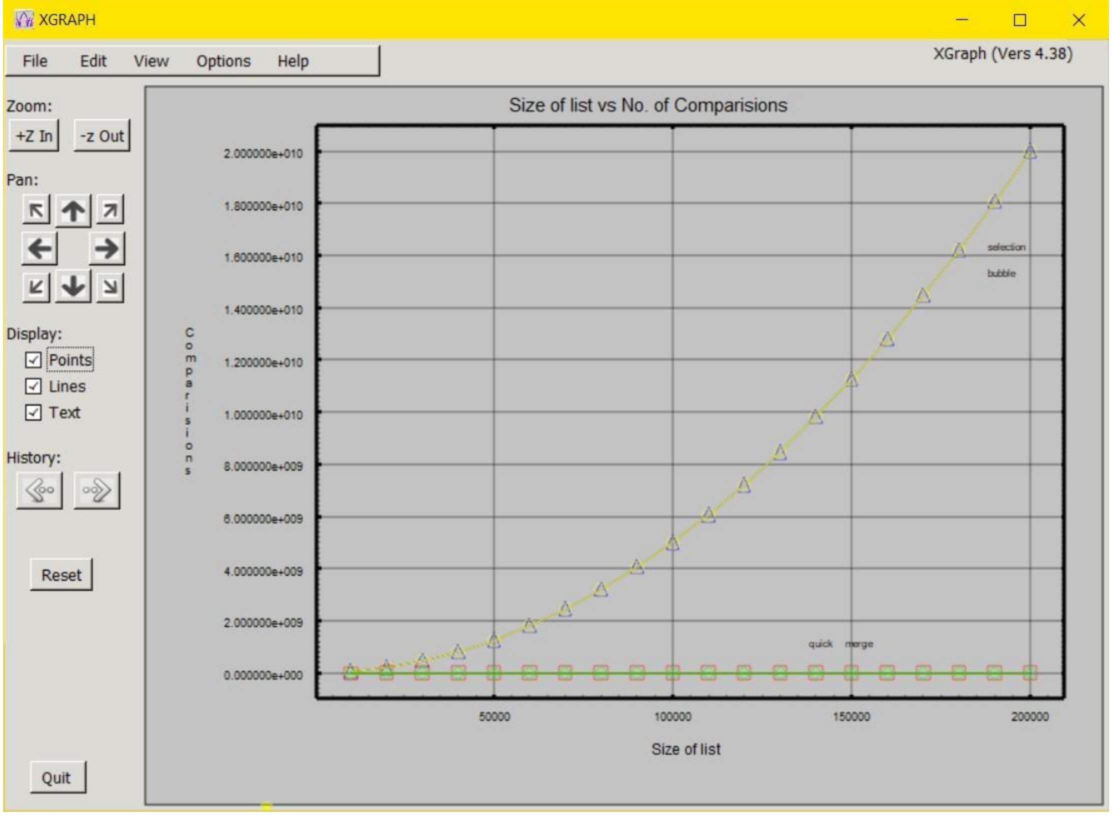
Merge sort



```
next
color = green
10000 140474
20000 300989
30000 468599
40000 641980
50000 818133
60000 997238
70000 1178982
80000 1363663
90000 1549513
100000 1736493
110000 1925220
120000 2114438
130000 2304463
140000 2498083
150000 2692697
160000 2886964
170000 3083115
180000 3279057
190000 3475065
200000 3672627

title_x = Size of list
title_y = Comparisions
Title = Size of list vs No. of Comparisions
anno 190000 16199910000 selection
anno 190000 15199910000 bubble
anno 140000 1000000000 quick
anno 150000 1000000000 merge
```

Graph Plot comparing different algorithms

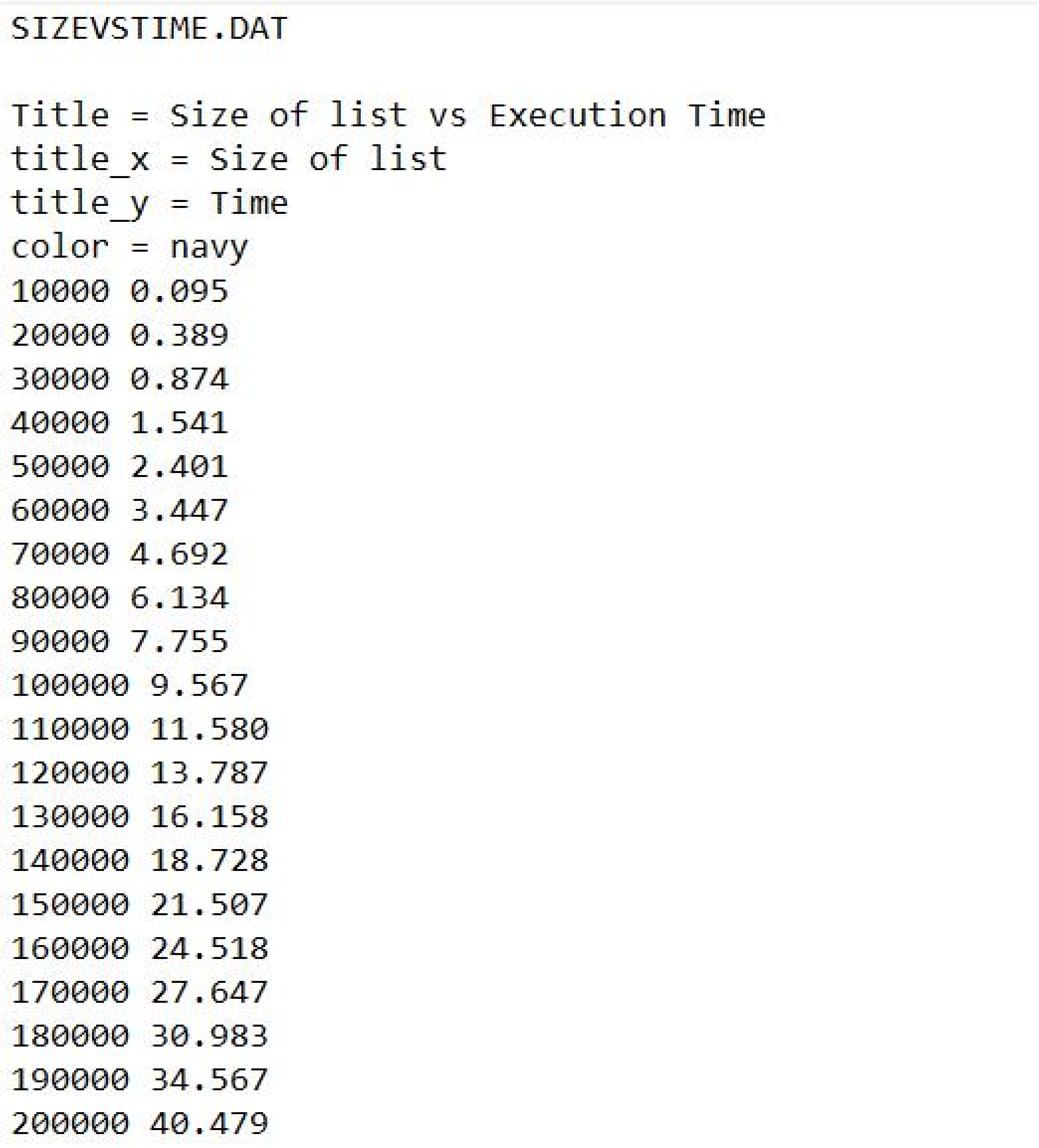
(Size of list v/s Comparision time)



Display:
☑ Points
☑ Lines
☑ Text

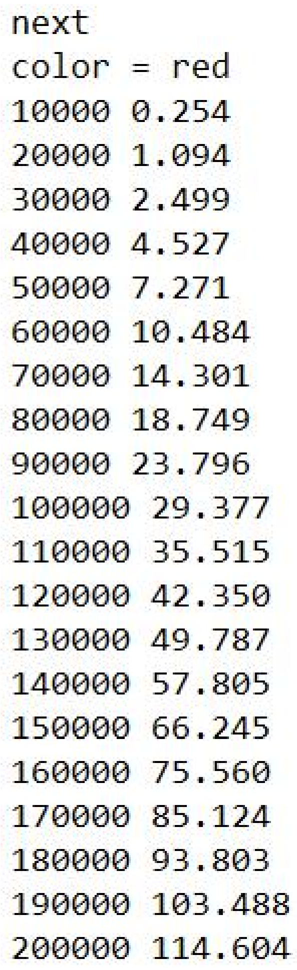
Size of list v/s Execution time

Selection sort



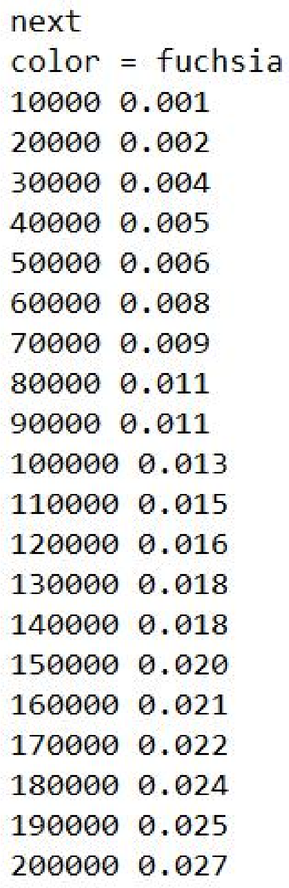
```
SIZEVSTIME.DAT

Title = Size of list vs Execution Time
title_x = Size of list
title_y = Time
color = navy
10000 0.095
20000 0.389
30000 0.874
40000 1.541
50000 2.401
60000 3.447
70000 4.692
80000 6.134
90000 7.755
100000 9.567
110000 11.580
120000 13.787
130000 16.158
140000 18.728
150000 21.507
160000 24.518
170000 27.647
180000 30.983
190000 34.567
200000 40.479
```

Bubble sort



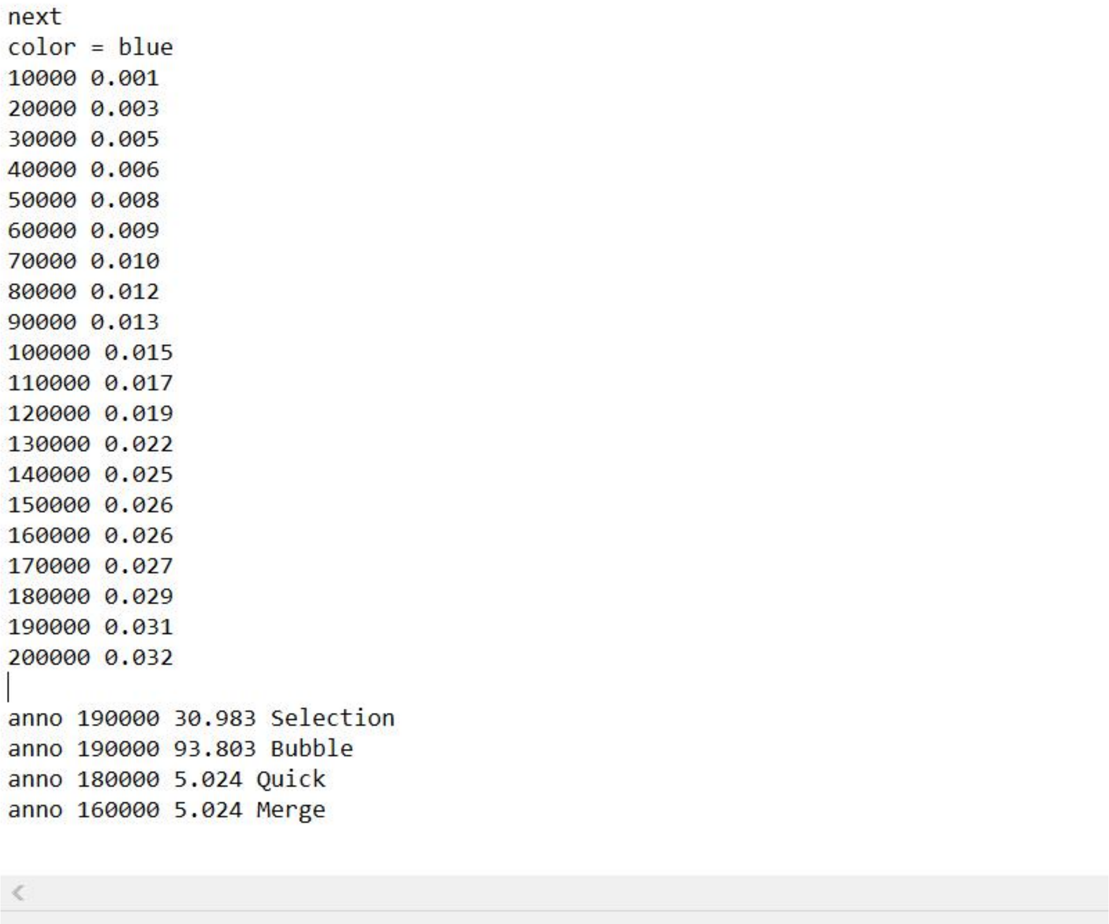
```
next
color = red
10000 0.254
20000 1.094
30000 2.499
40000 4.527
50000 7.271
60000 10.484
70000 14.301
80000 18.749
90000 23.796
100000 29.377
110000 35.515
120000 42.350
130000 49.787
140000 57.805
150000 66.245
160000 75.560
170000 85.124
180000 93.803
190000 103.488
200000 114.604
```

Qucik sort



```
next
color = fuchsia
10000 0.001
20000 0.002
30000 0.004
40000 0.005
50000 0.006
60000 0.008
70000 0.009
80000 0.011
90000 0.011
100000 0.013
110000 0.015
120000 0.016
130000 0.018
140000 0.018
150000 0.020
160000 0.021
170000 0.022
180000 0.024
190000 0.025
200000 0.027
```

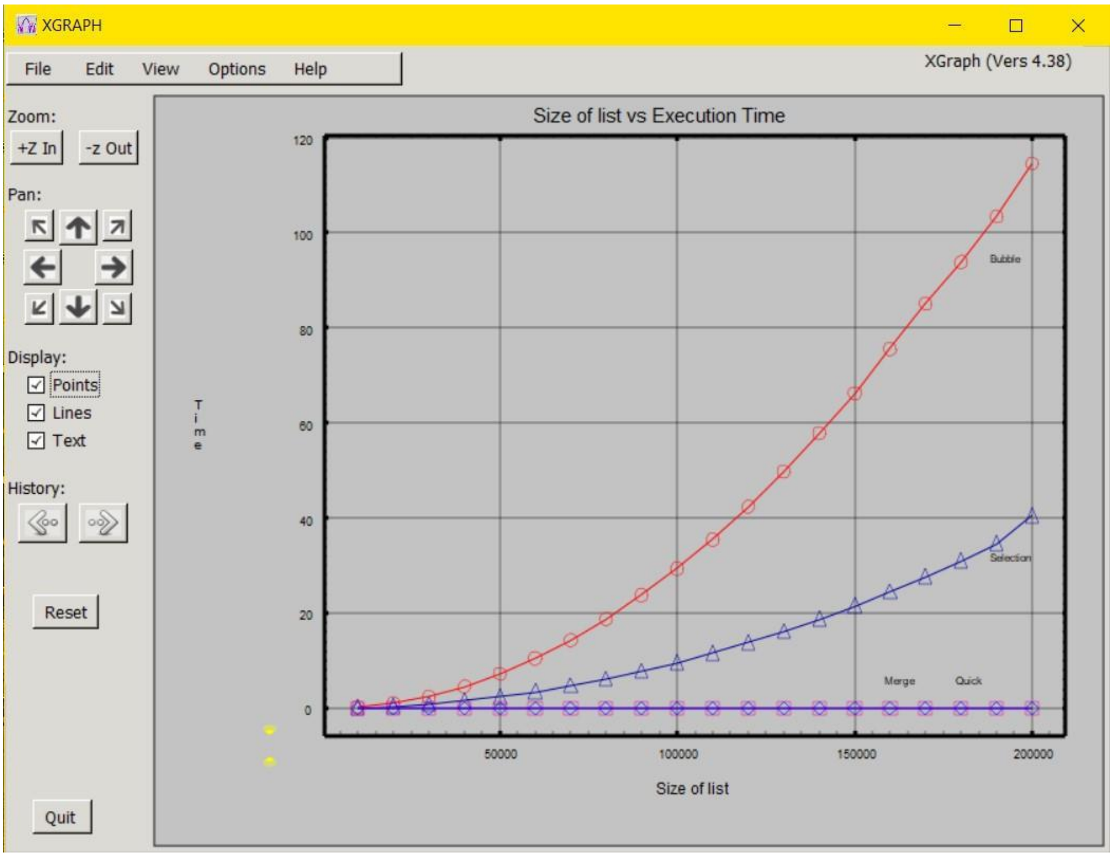
Merge sort



```
next
color = blue
10000 0.001
20000 0.003
30000 0.005
40000 0.006
50000 0.008
60000 0.009
70000 0.010
80000 0.012
90000 0.013
100000 0.015
110000 0.017
120000 0.019
130000 0.022
140000 0.025
150000 0.026
160000 0.026
170000 0.027
180000 0.029
190000 0.031
200000 0.032
|
anno 190000 30.983 Selection
anno 190000 93.803 Bubble
anno 180000 5.024 Quick
anno 160000 5.024 Merge
```

Graph Plot comparing different algorithms

(Size of list v/s Execution time)



***