UM-SJTU JOINT INSTITUTE
Data Structures and Algorithms
(VP281)

Programming Assignment

Programming Assignment One
Sorting

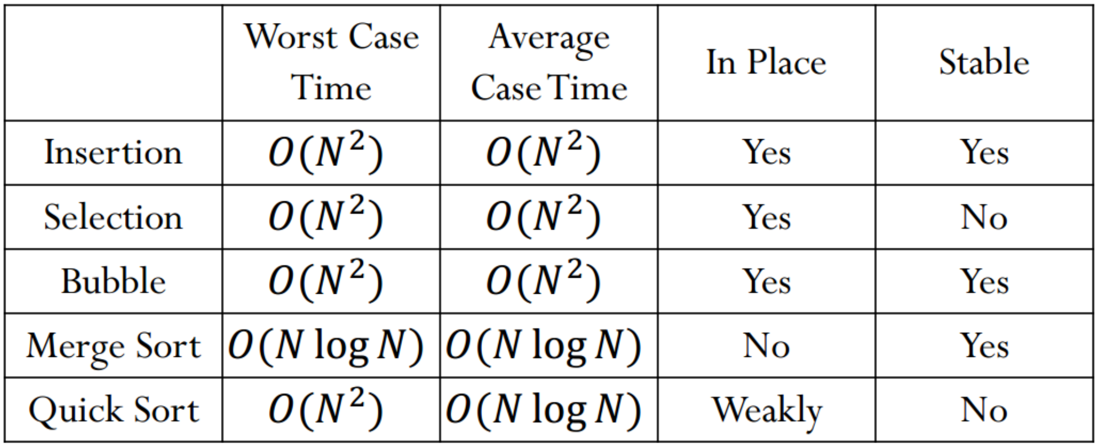
Name: Pan Chongdan
ID: 516370910121

Date: September 28, 2018

1 Introduction

The programming assignment asks me to implement six sorting algorithms including bubble sort, insertion sort, selection sort, merge sort and quick sort with in-place and not-inplace. The goal is clear, to gain experience in implementing these six sorting algorithms to sort lots of random numbers in ascending order.

Second, the assignment asks me to study the performance of these 6 studies by studying their time efficiency. In lecture slides3, professor gives a table summarizing the time required for each algorithm to complete the sorting process for consideration.



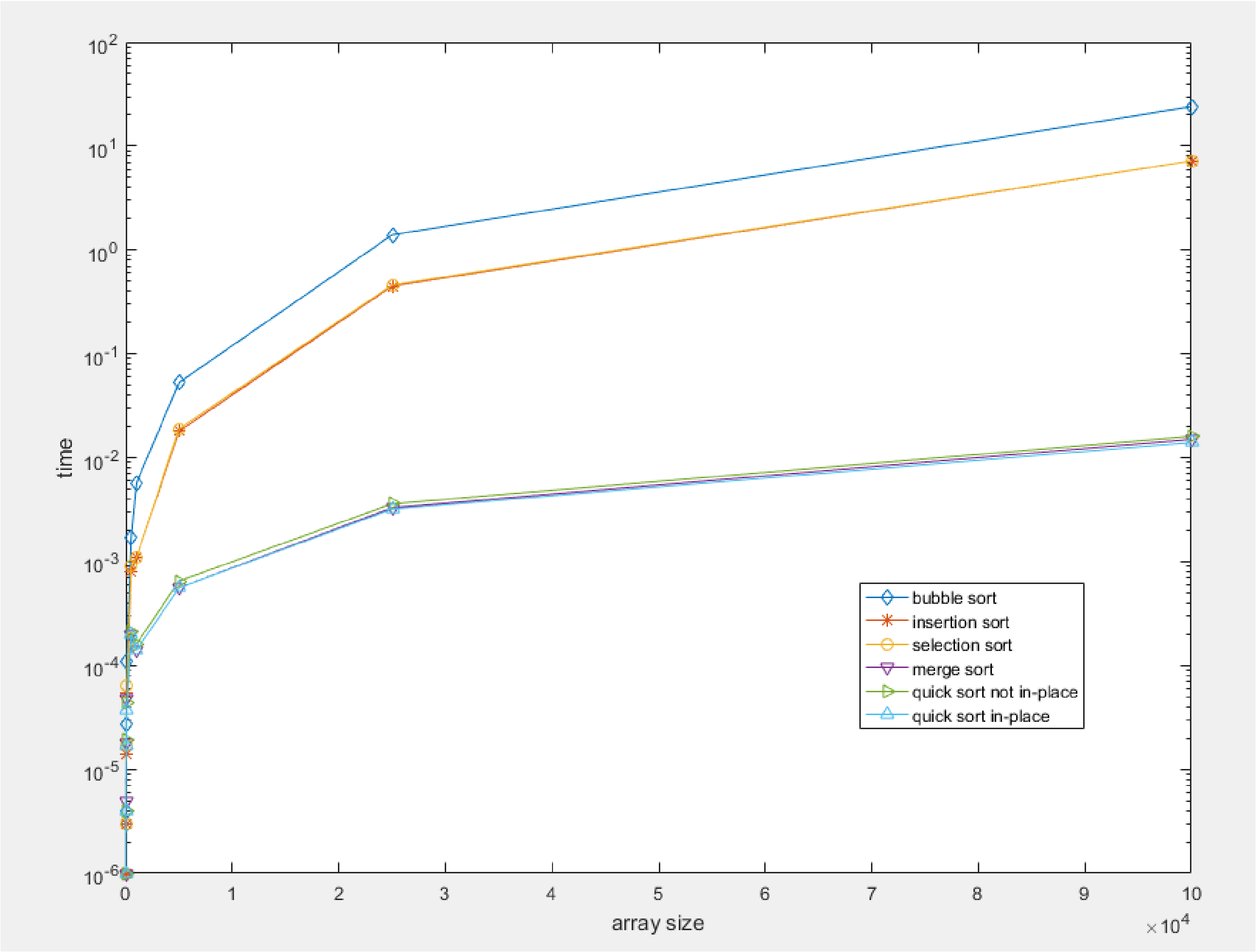
	Worst Case Time	Average Case Time	In Place	Stable
Insertion	$O(N^2)$	$O(N^2)$	Yes	Yes
Selection	$O(N^2)$	$O(N^2)$	Yes	No
Bubble	$O(N^2)$	$O(N^2)$	Yes	Yes
Merge Sort	$O(N \log N)$	$O(N \log N)$	No	Yes
Quick Sort	$O(N^2)$	$O(N \log N)$	Weakly	No

However, we need to test the algorithms' time efficiency by ourselves, so I wrote a cpp file to print out the time required for each algorithm, which is included in appendix part.

2 Result

To test time efficiency, I used clock() function to get the time required. To get rid of other disturbing factor, I used to variable start and stop to get the time right before and after the sorting complete, then their difference is the time used. In my analysis, I get 10 set of data ,from which the sorted array' size is 1, 10, 50, 100, 500, 1000, 5000, 25000 and 100000.

Data Size	1	10	50	100	500	1000	5000	25000	100000
Bubble Sort	3e-6	4e-6	2.7e-5	1.1e-4	1.7e-3	0.0057	0.053	1.4	24
Insertion Sort	1e-6	3e-6	1.4e-5	5.4e-5	8.1e-4	0.0011	0.018	0.45	7.2
Selection Sort	1e-6	3e-6	1.7e-5	6.3e-5	9e-4	0.0011	0.019	0.46	7.2
Merge Sort	1e-6	5e-6	1.8e-5	4.8e-5	2e-4	1.4e-4	5.6e-4	0.0033	0.015
Quick Sort (not in-place)	1e-6	4e-6	1.9e-5	4.3e-5	2.1e-4	1.6e-4	6.5e-4	0.0036	0.016
Quick Sort (in place)	1e-6	4e-6	1.7e-5	3.7e-5	2e-4	1.4e-4	5.6e-4	0.0032	0.014



3 Conclusion

This chart is plot by matlab, showing the time efficiency of each algorithms compared to each other, and it has following characteristics.

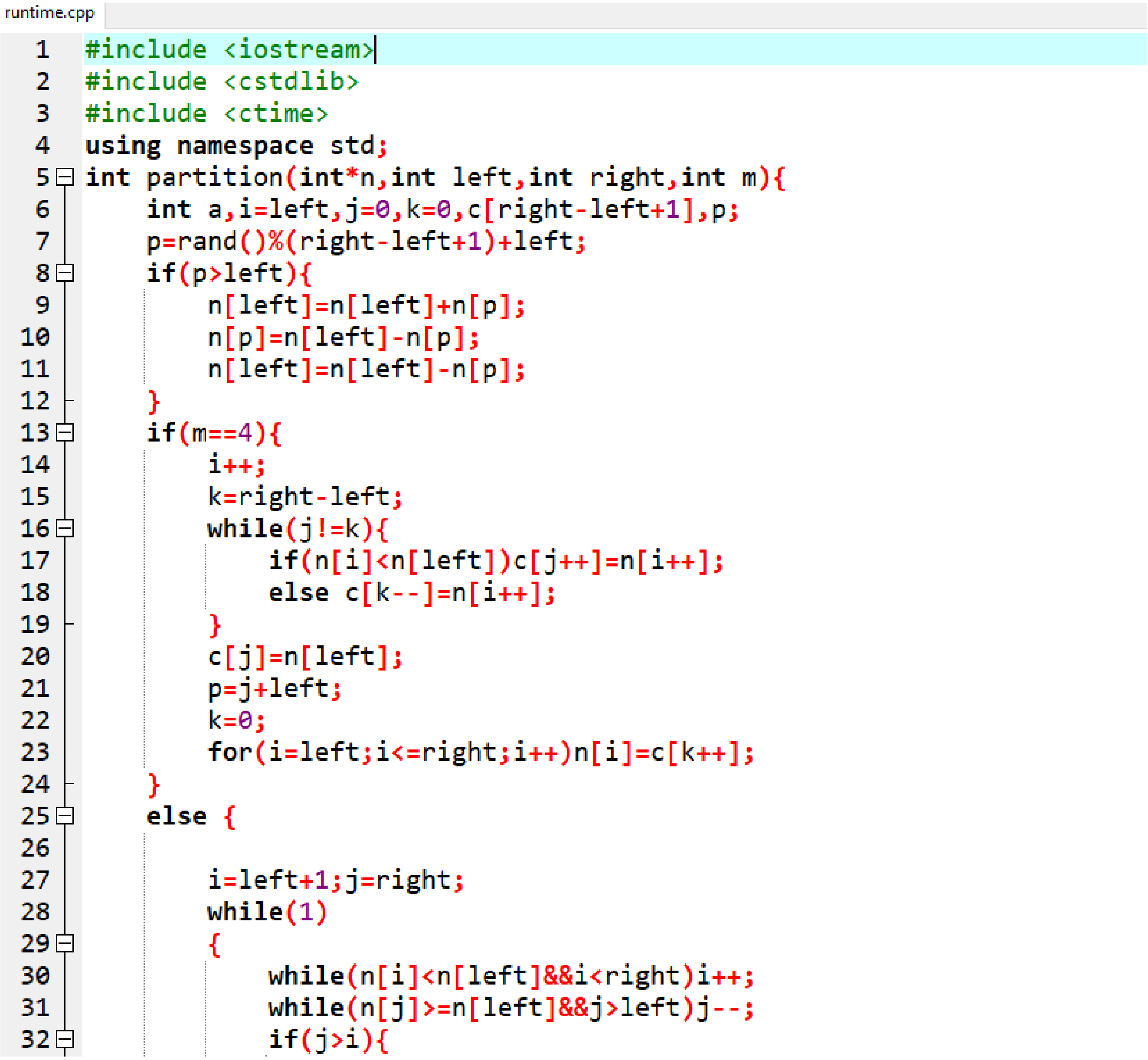
1. The average time required for each algorithm grows as the array grows bigger.

2. As shown in the graph before, bubble sort, insertion sort and selection sort require much more time to sort huge array than the other sort algorithms. However, when the data size is extremely small, there is no such characteristic and the insertion sort and selection sort even requires less time than merge sort and quick sort.

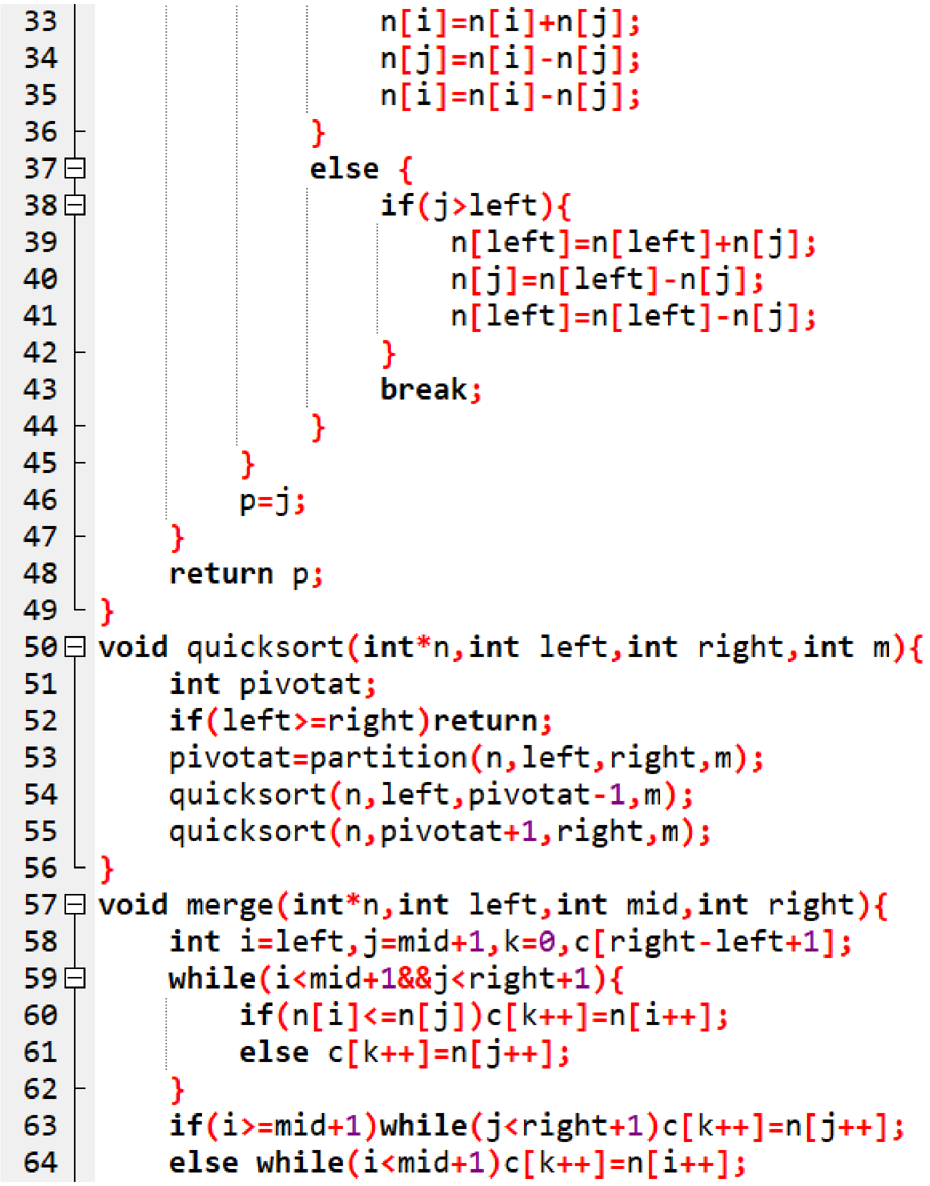
3. Even though insertion sort, selection sort and bubble sort have the same time complexity as shown in the graph from lecture, it seems that bubble sort needs more time to sort than others. I guess its because bubble sort needs to compare every two members of the array once.

4 Appendix

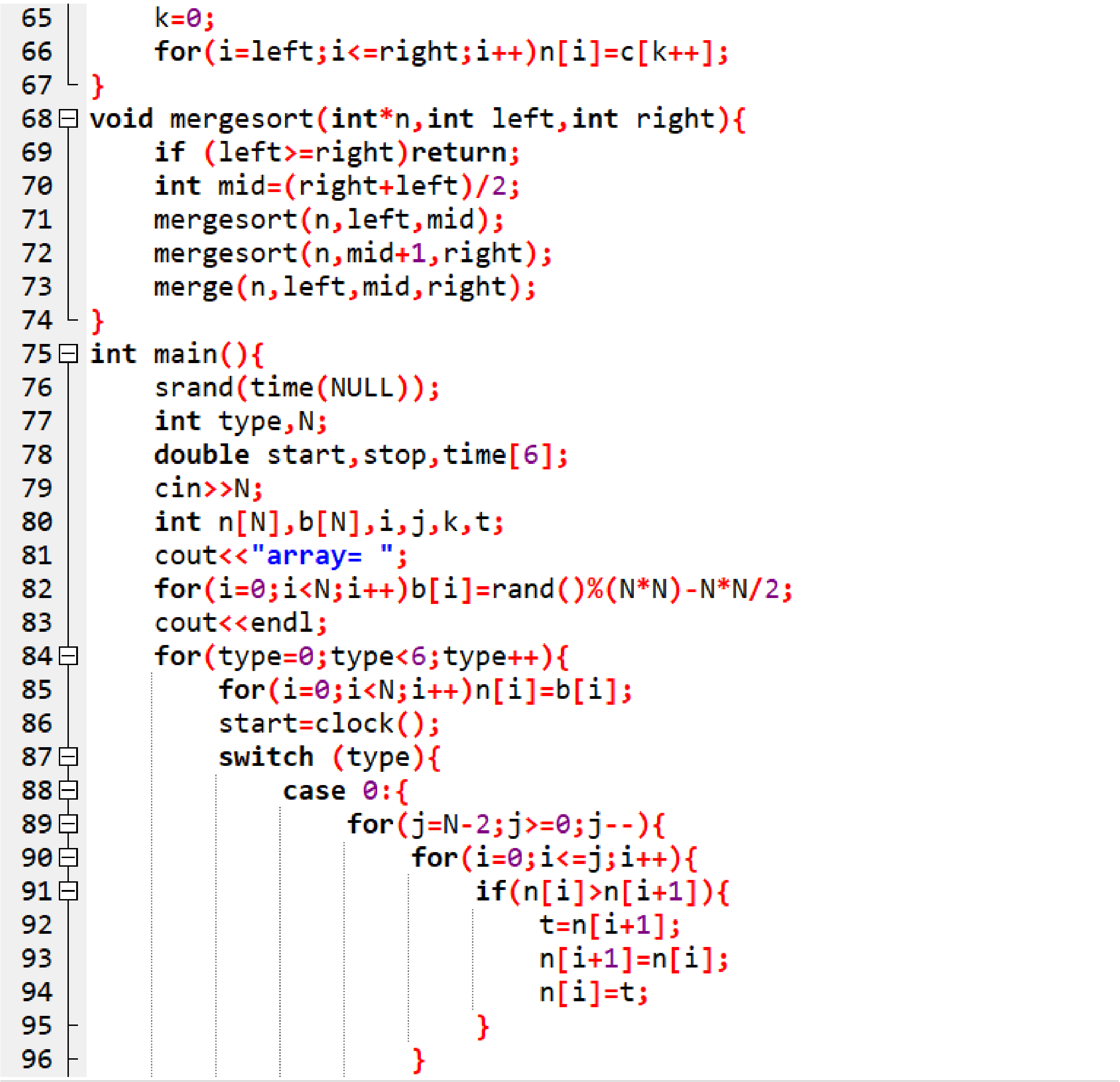
The appendix shows the cpp code for time comparison.



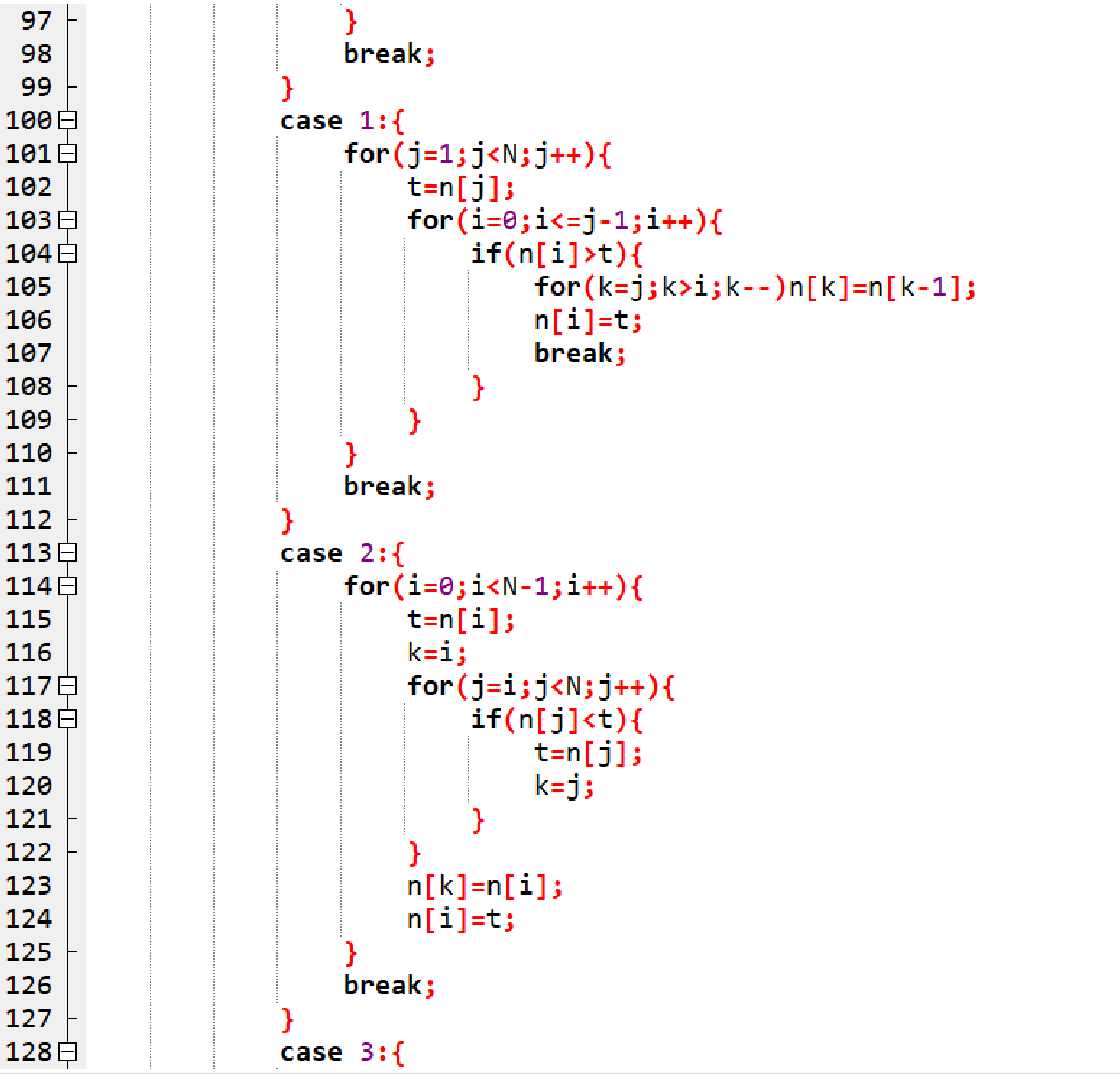
```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
int partition(int*n,int left,int right,int m){
    int a,i=left,j=0,k=0,c[right-left+1],p;
    p=rand()%(right-left+1)+left;
    if(p>left){
        n[left]=n[left]+n[p];
        n[p]=n[left]-n[p];
        n[left]=n[left]-n[p];
    }
    if(m==4){
        i++;
        k=right-left;
        while(j!=k){
            if(n[i]<n[left])c[j++]=n[i++];
            else c[k--]=n[i++];
        }
        c[j]=n[left];
        p=j+left;
        k=0;
        for(i=left;i<=right;i++)n[i]=c[k++];
    }
    else {

        i=left+1;j=right;
        while(1)
        {
            while(n[i]<n[left]&&i<right)i++;
            while(n[j]>=n[left]&&j>left)j--;
            if(j>i){
```



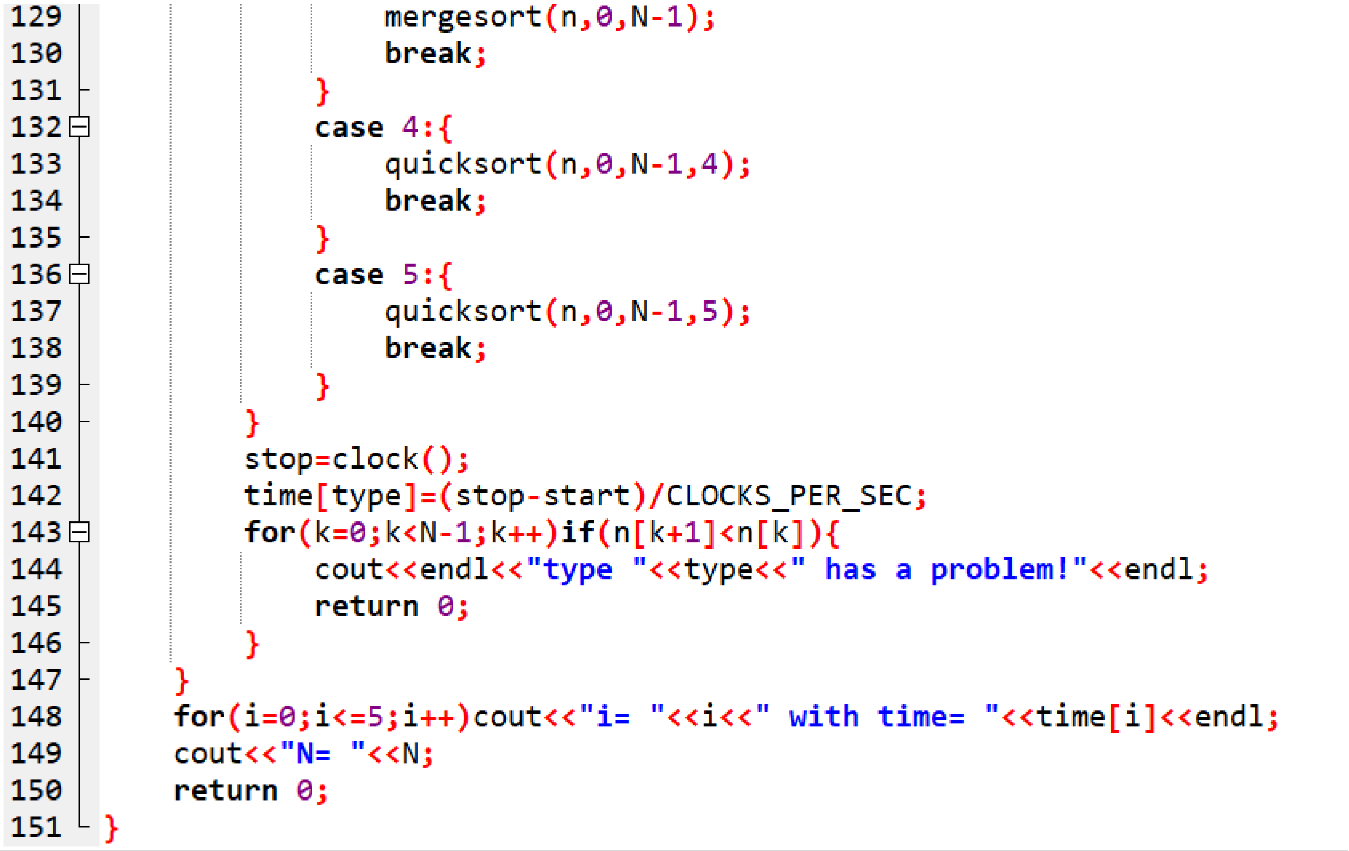
```
                n[i]=n[i]+n[j];
                n[j]=n[i]-n[j];
                n[i]=n[i]-n[j];
            }
            else {
                if(j>left){
                    n[left]=n[left]+n[j];
                    n[j]=n[left]-n[j];
                    n[left]=n[left]-n[j];
                }
                break;
            }
        }
        p=j;
    }
    return p;
}
void quicksort(int*n,int left,int right,int m){
    int pivotat;
    if(left>=right)return;
    pivotat=partition(n,left,right,m);
    quicksort(n,left,pivotat-1,m);
    quicksort(n,pivotat+1,right,m);
}
void merge(int*n,int left,int mid,int right){
    int i=left,j=mid+1,k=0,c[right-left+1];
    while(i<mid+1&&j<right+1){
        if(n[i]<=n[j])c[k++]=n[i++];
        else c[k++]=n[j++];
    }
    if(i>=mid+1)while(j<right+1)c[k++]=n[j++];
    else while(i<mid+1)c[k++]=n[i++];
```



```
    k=0;
    for(i=left;i<=right;i++)n[i]=c[k++];
}
void mergesort(int*n,int left,int right){
    if (left>=right)return;
    int mid=(right+left)/2;
    mergesort(n,left,mid);
    mergesort(n,mid+1,right);
    merge(n,left,mid,right);
}
int main(){
    srand(time(NULL));
    int type,N;
    double start,stop,time[6];
    cin>>N;
    int n[N],b[N],i,j,k,t;
    cout<<"array= ";
    for(i=0;i<N;i++)b[i]=rand()%(N*N)-N*N/2;
    cout<<endl;
    for(type=0;type<6;type++){
        for(i=0;i<N;i++)n[i]=b[i];
        start=clock();
        switch (type){
            case 0:{
                for(j=N-2;j>=0;j--){
                    for(i=0;i<=j;i++){
                        if(n[i]>n[i+1]){
                            t=n[i+1];
                            n[i+1]=n[i];
                            n[i]=t;
                        }
                    }
```

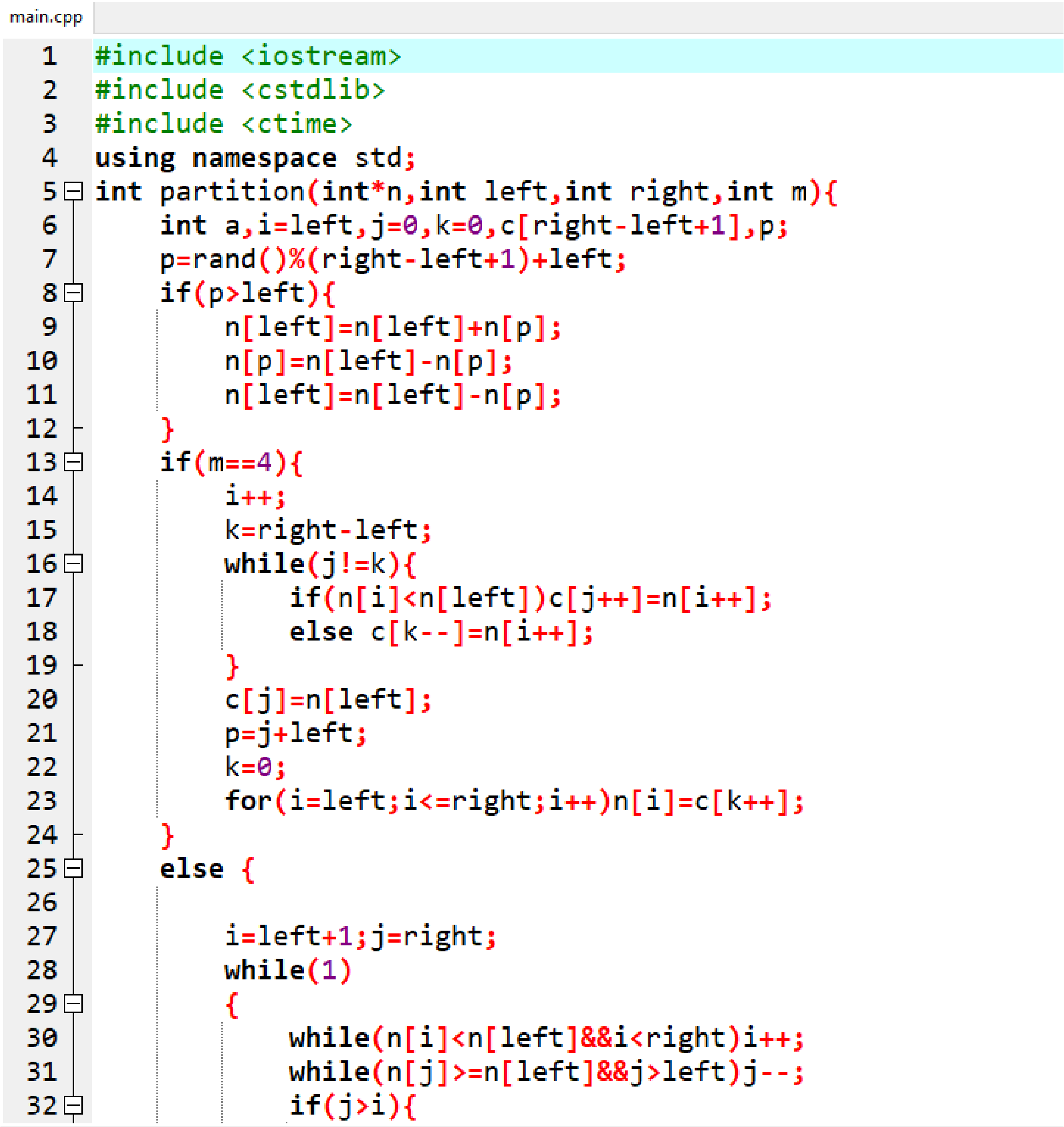


```
                }
                break;
            }
            case 1:{
                for(j=1;j<N;j++){
                    t=n[j];
                    for(i=0;i<=j-1;i++){
                        if(n[i]>t){
                            for(k=j;k>i;k--)n[k]=n[k-1];
                            n[i]=t;
                            break;
                        }
                    }
                }
                break;
            }
            case 2:{
                for(i=0;i<N-1;i++){
                    t=n[i];
                    k=i;
                    for(j=i;j<N;j++){
                        if(n[j]<t){
                            t=n[j];
                            k=j;
                        }
                    }
                    n[k]=n[i];
                    n[i]=t;
                }
                break;
            }
            case 3:{
```

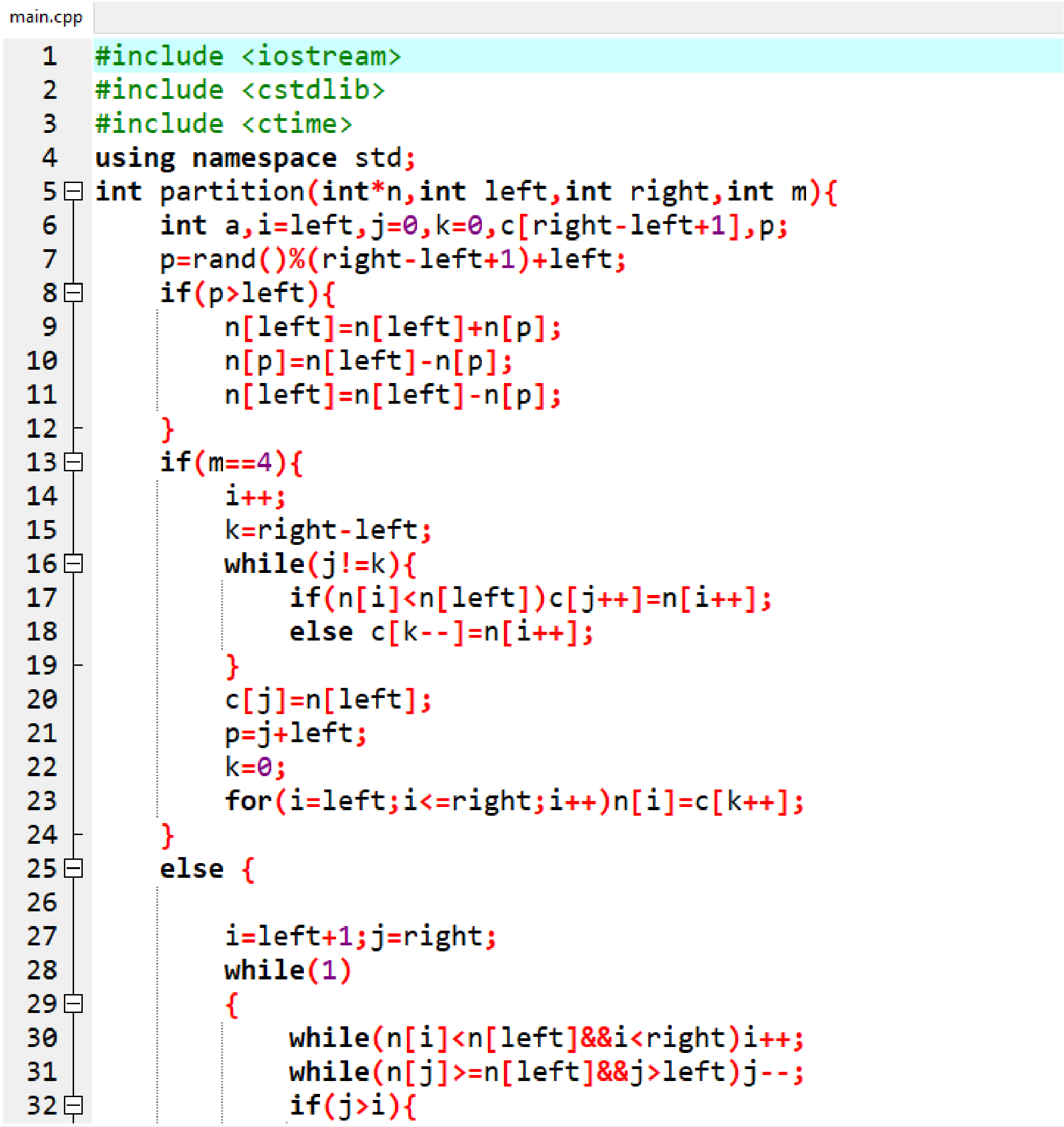


```
                mergesort(n,0,N-1);
                break;
            }
            case 4:{
                quicksort(n,0,N-1,4);
                break;
            }
            case 5:{
                quicksort(n,0,N-1,5);
                break;
            }
        }
        stop=clock();
        time[type]=(stop-start)/CLOCKS_PER_SEC;
        for(k=0;k<N-1;k++)if(n[k+1]<n[k]){
            cout<<endl<<"type "<<type<<" has a problem!"<<endl;
            return 0;
        }
    }
    for(i=0;i<=5;i++)cout<<"i= "<<i<<" with time= "<<time[i]<<endl;
    cout<<"N= "<<N;
    return 0;
}
```

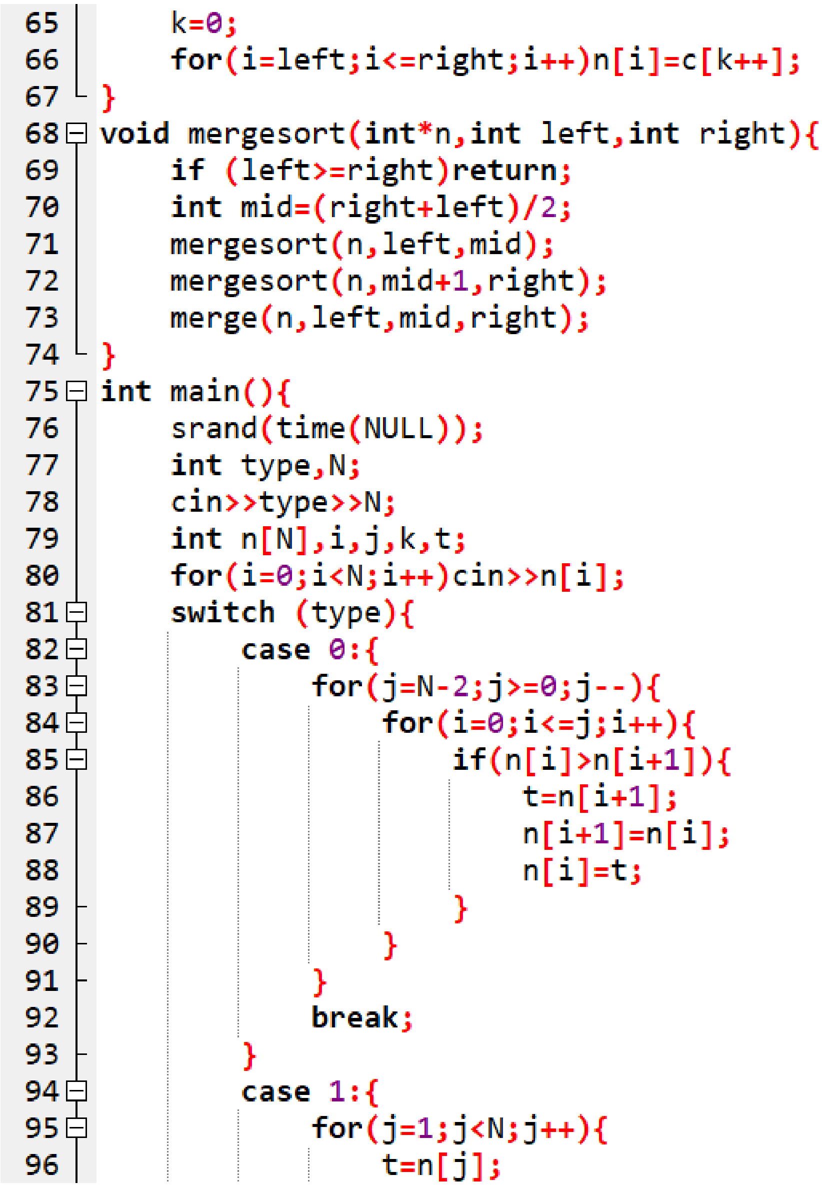
Then the appendix shows the cpp code for each sort algorithm



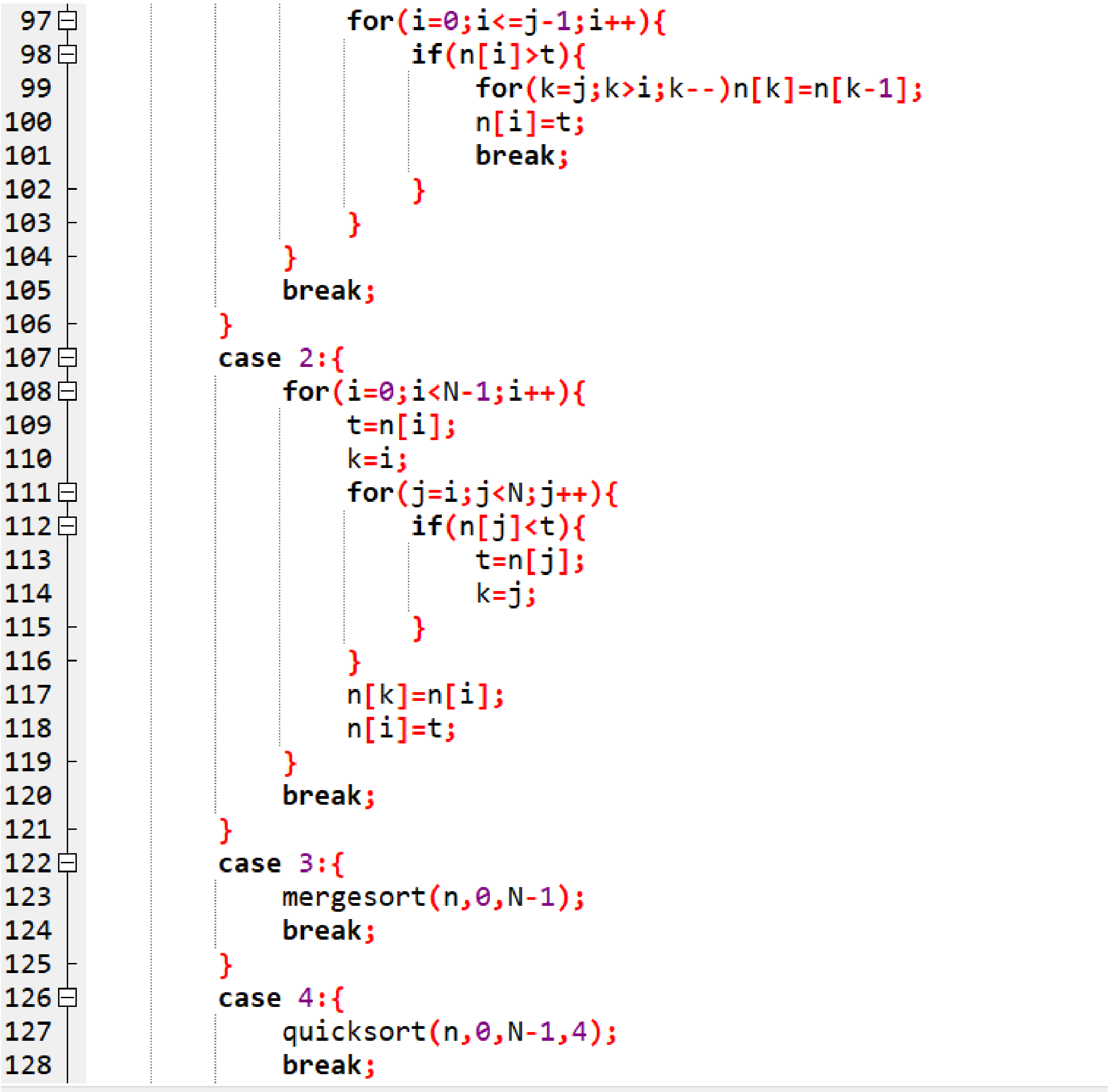
```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
int partition(int*n,int left,int right,int m){
    int a,i=left,j=0,k=0,c[right-left+1],p;
    p=rand()%(right-left+1)+left;
    if(p>left){
        n[left]=n[left]+n[p];
        n[p]=n[left]-n[p];
        n[left]=n[left]-n[p];
    }
    if(m==4){
        i++;
        k=right-left;
        while(j!=k){
            if(n[i]<n[left])c[j++]=n[i++];
            else c[k--]=n[i++];
        }
        c[j]=n[left];
        p=j+left;
        k=0;
        for(i=left;i<=right;i++)n[i]=c[k++];
    }
    else {

        i=left+1;j=right;
        while(1)
        {
            while(n[i]<n[left]&&i<right)i++;
            while(n[j]>=n[left]&&j>left)j--;
            if(j>i){
```



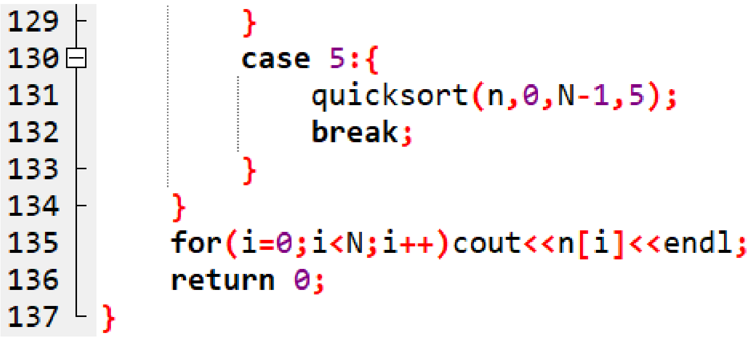
```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
int partition(int*n,int left,int right,int m){
    int a,i=left,j=0,k=0,c[right-left+1],p;
    p=rand()%(right-left+1)+left;
    if(p>left){
        n[left]=n[left]+n[p];
        n[p]=n[left]-n[p];
        n[left]=n[left]-n[p];
    }
    if(m==4){
        i++;
        k=right-left;
        while(j!=k){
            if(n[i]<n[left])c[j++]=n[i++];
            else c[k--]=n[i++];
        }
        c[j]=n[left];
        p=j+left;
        k=0;
        for(i=left;i<=right;i++)n[i]=c[k++];
    }
    else {

        i=left+1;j=right;
        while(1)
        {
            while(n[i]<n[left]&&i<right)i++;
            while(n[j]>=n[left]&&j>left)j--;
            if(j>i){
```



```
    k=0;
    for(i=left;i<=right;i++)n[i]=c[k++];
}
void mergesort(int*n,int left,int right){
    if (left>=right)return;
    int mid=(right+left)/2;
    mergesort(n,left,mid);
    mergesort(n,mid+1,right);
    merge(n,left,mid,right);
}
int main(){
    srand(time(NULL));
    int type,N;
    cin>>type>>N;
    int n[N],i,j,k,t;
    for(i=0;i<N;i++)cin>>n[i];
    switch (type){
        case 0:{
            for(j=N-2;j>=0;j--){
                for(i=0;i<=j;i++){
                    if(n[i]>n[i+1]){
                        t=n[i+1];
                        n[i+1]=n[i];
                        n[i]=t;
                    }
                }
            }
            break;
        }
        case 1:{
            for(j=1;j<N;j++){
                t=n[j];
```



```
                for(i=0;i<=j-1;i++){
                    if(n[i]>t){
                        for(k=j;k>i;k--)n[k]=n[k-1];
                        n[i]=t;
                        break;
                    }
                }
            }
            break;
        }
        case 2:{
            for(i=0;i<N-1;i++){
                t=n[i];
                k=i;
                for(j=i;j<N;j++){
                    if(n[j]<t){
                        t=n[j];
                        k=j;
                    }
                }
                n[k]=n[i];
                n[i]=t;
            }
            break;
        }
        case 3:{
            mergesort(n,0,N-1);
            break;
        }
        case 4:{
            quicksort(n,0,N-1,4);
            break;
```



```
        }
        case 5:{
            quicksort(n,0,N-1,5);
            break;
        }
    }
    for(i=0;i<N;i++)cout<<n[i]<<endl;
    return 0;
}
```