**Sorting Algorithms**

**1)Bubble sort**

#include <iostream>

#include<stdio.h>

#include<conio.h>

#include <stdlib.h>

#include<time.h>

using namespace std;

void bubbleSort(int array[], int size)

{

for (int step = 0; step < size - 1; ++step)

{

for (int i = 0; i < size - step - 1; ++i)

{

if (array[i] > array[i + 1])

{

int temp = array[i];

array[i] = array[i + 1];

array[i + 1] = temp;

}

}

}

}

void printArray(int array[], int size)

{

for (int i = 0; i < size; ++i)

{

cout << " " << array[i];

}

cout << "\n";

}

int main()

{

int n;

printf("Enter number of elements\n");

scanf("%d", &n);

printf("%d integers are\n",n);

srand(time(0));

int i;

int arr[n];

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

{

arr[i]=rand()%5000+1000;

printf("%d\n",arr[i]);

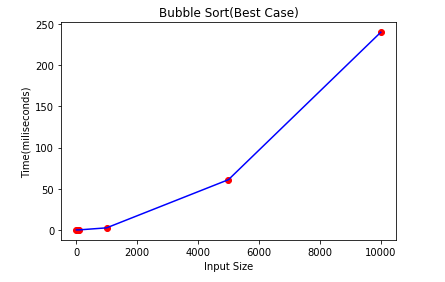
}

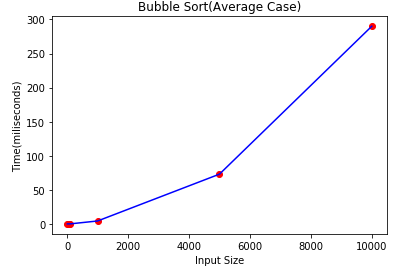
bubbleSort(arr,n);

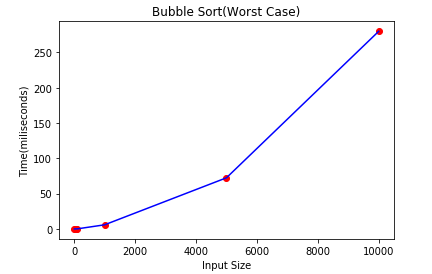
cout <<"Sorted Array is:\n";

printArray(arr,n);

}







**2)External merge sort**

#include <bits/stdc++.h>

#include<iostream>

#include<stdio.h>

#include<conio.h>

#include <stdlib.h>

#include<time.h>

using namespace std;

struct Min\_heap

{

int e;

int i;

};

struct comp

{

bool operator()(const Min\_heap l, const Min\_heap r) const

{

return l.e > r.e;

}

};

FILE\* openFile(char\* fName, char\* mode)

{

FILE\* fp = fopen(fName, mode);

if (fp == NULL)

{

perror("Error while opening the file.\n");

exit(EXIT\_FAILURE);

}

return fp;

}

void mergeFiles(char \*output\_file, int n, int k)

{

FILE\* in[k];

for (int i = 0; i < k; i++)

{

char fName[2];

snprintf(fName, sizeof(fName), "%d", i);

in[i] = openFile(fName, "r");

}

FILE \*out = openFile(output\_file, "w");

Min\_heap h[k];

priority\_queue<Min\_heap, std::vector<Min\_heap>, comp> pq;

int i;

for (i = 0; i < k; i++)

{

if (fscanf(in[i], "%d ", &h[i].e) != 1)

break;

h[i].i = i;

pq.push(h[i]);

}

int count = 0;

while (count != i)

{

Min\_heap root = pq.top();

pq.pop();

fprintf(out, "%d ", root.e);

if (fscanf(in[root.i], "%d ", &root.e) != 1 )

{

root.e = INT\_MAX;

count++;

}

pq.push(root);

}

for (int i = 0; i < k; i++)

fclose(in[i]);

fclose(out);

}

void init\_run(char \*input\_file, int run\_size, int n)

{

FILE \*in = openFile(input\_file, "r");

FILE\* out[n];

char fName[2];

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

{

snprintf(fName, sizeof(fName), "%d", i);

out[i] = openFile(fName, "w");

}

int\* arr = (int\*)malloc(run\_size \* sizeof(int));

bool more\_input = true;

int next\_output\_file = 0;

int i;

while (more\_input)

{

for (i = 0; i < run\_size; i++)

{

if (fscanf(in, "%d ", &arr[i]) != 1)

{

more\_input = false;

break;

}

}

sort(arr, arr + i);

for (int j = 0; j < i; j++)

fprintf(out[next\_output\_file], "%d ", arr[j]);

next\_output\_file++;

}

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

fclose(out[i]);

fclose(in);

}

int main()

{

int n = 10;

int run\_size = 1000;

char input\_file[] = "input.txt";

char output\_file[] = "output.txt";

FILE\* in = openFile(input\_file, "w");

srand(time(NULL));

for (int i = 0; i < n \* run\_size; i++)

fprintf(in, "%d ", rand());

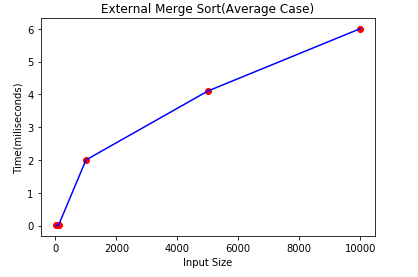
fclose(in);

init\_run(input\_file, run\_size, n);

mergeFiles(output\_file, run\_size, n);

return 0;

}



**3)Insertion sort**

#include <stdio.h>

#include <stdlib.h>

#include<time.h>

int main()

{

int n,arr[1000],i,d,t,flag=0;

printf("Enter number of elements\n");

scanf("%d", &n);

printf("Enter %d integers\n",n);

srand(time(0));

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

{

arr[i]=rand()%5000;

printf("%d\n",arr[i]);

}

for (i=1;i <= n-1;i++)

{t = arr[i];

for (d=i-1;i>=0;i--)

{

if (arr[d]>t)

{

arr[d+1]=arr[d];

flag = 1;

}

else

{break;}

}

if (flag)

{arr[d+1] = t;}

}

printf("\n\nSorted list\n");

for (i=0;i<= n - 1;i++)

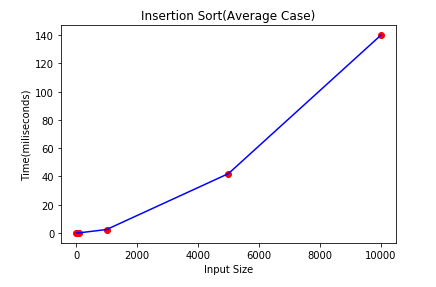
{

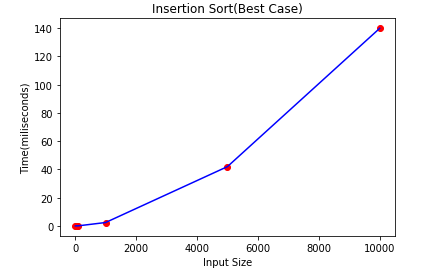
printf("%d\n", arr[i]);

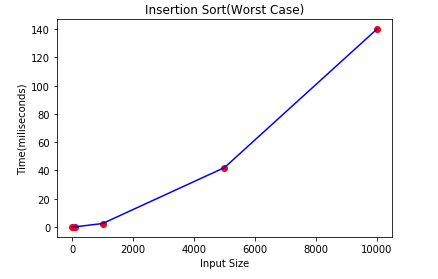
}

return 0;

}







4)**Merge Sort**

#include<stdlib.h>

#include<stdio.h>

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1+ j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2)

{

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 mergeSort(int arr[], int l, int r)

{

if (l < r)

{

int m = l+(r-l);

mergeSort(arr, l, m);

mergeSort(arr, m+1, r);

merge(arr, l, m, r);

}

}

void printArray(int A[], int size)

{

int i;

for (i=0; i < size; i++)

printf("%d ", A[i]);

printf("\n");

}

int main()

{

int arr[] = {12, 11, 13, 5, 6, 7};

int arr\_size = sizeof(arr)/sizeof(arr[0]);

printf("Given array is \n");

printArray(arr, arr\_size);

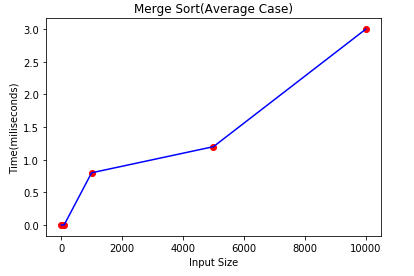
mergeSort(arr, 0, arr\_size - 1);

printf("\nSorted array is \n");

printArray(arr, arr\_size);

return 0;

}



5) **Quick sort**

#include <bits/stdc++.h>

#include<iostream>

#include<stdio.h>

#include<conio.h>

#include <stdlib.h>

#include<time.h>

using namespace std;

void swap\_var(int\* a, int\* b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

int partition (int arr[], int l, int h)

{

int pivot = arr[h];

int i = (l - 1);

for (int j = l; j <= h - 1; j++)

{

if (arr[j] < pivot)

{

i++;

swap\_var(&arr[i], &arr[j]);

}

}

swap\_var(&arr[i + 1], &arr[h]);

return (i + 1);

}

void quickSort(int arr[], int l, int h)

{

if (l < h)

{

int p = partition(arr, l, h);

quickSort(arr, l, p - 1);

quickSort(arr, p + 1, h);

}

}

int main()

{

int n;

printf("Enter number of elements\n");

scanf("%d", &n);

printf("%d integers are\n",n);

srand(time(0));

int i;

int arr[n];

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

{

arr[i]=rand()%5000+1000;

printf("%d\n",arr[i]);

}

quickSort(arr, 0, n - 1);

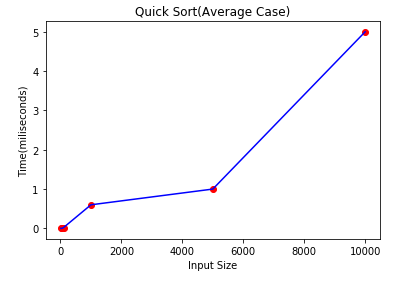
cout << "Sorted array is: \n";

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

cout << arr[i] << " ";

return 0;

}

****

**6)Selection sort**

#include <stdio.h>

#include<conio.h>

void swap(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

void selectionSort(int arr[], int n)

{

int i, j, min\_i;

for (i = 0; i < n-1; i++)

{

min\_i = i;

for (j = i+1; j < n; j++)

if (arr[j] < arr[min\_i])

min\_i = j;

swap(&arr[min\_i], &arr[i]);

}

}

int main()

{

int n;

printf("Enter number of elements\n");

scanf("%d", &n);

printf("Enter %d integers\n",n);

srand(time(0));

int i;

int arr[n];

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

{

arr[i]=rand()%5000+1000;

printf("%d\n",arr[i]);

}

selectionSort(arr, n);

cout << "Sorted array: \n";

int i;

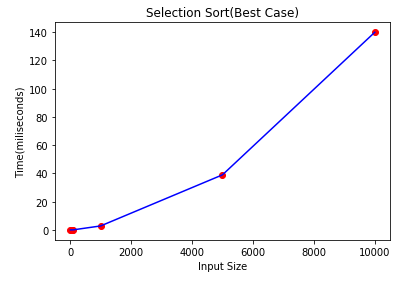
for (i=0; i < size; i++)

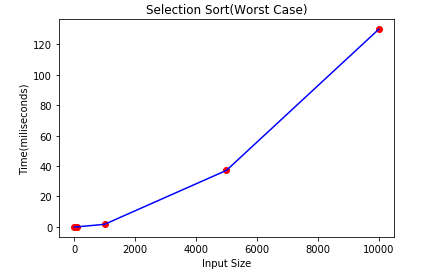
cout << arr[i] << " ";

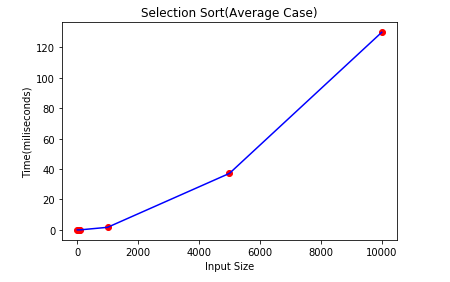
cout << endl;

return0**;**

**}**

****

****

****

**7)Bitonic Sort**

#include <stdio.h>

#include <stdlib.h>

int up = 1;

int down = 0;

void compare(int i, int j, int k,int arr[])

{

int t;

if (k == (arr[i] > arr[j]))

{

t=arr[i];

arr[i]=arr[j];

arr[j]=t;

}

}

void bitonicmerge(int low, int c, int dir,int arr[])

{

int k, i;

if (c > 1)

{

k = c / 2;

for (i = low;i < low+k ;i++)

compare(i, i+k, dir,arr);

bitonicmerge(low, k, dir,arr);

bitonicmerge(low+k, k, dir,arr);

}

}

void recbitonic(int low, int c, int dir,int arr[])

{

int k;

if (c > 1)

{

k = c / 2;

recbitonic(low, k, up,arr);

recbitonic(low + k, k, down,arr);

bitonicmerge(low, c, dir,arr);

}

}

void sort(int n,int arr[])

{

recbitonic(0,n,up,arr);

}

int main()

{

int i;

int n;

printf("Enter number of elements\n");

scanf("%d", &n);

printf("%d integers are\n",n);

srand(time(0));

int arr[n];

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

{

arr[i]=rand()%5000+1000;

printf("%d\n",arr[i]);

}

sort(n,arr);

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

{

printf("%d ", arr[i]);

}

return 0;

}