**Practical - 3**

Write user defined functions for the following sorting methods and compare their performance by time measurement with random data and Sorted data.

1. Selection Sort
2. Bubble Sort
3. Insertion Sort
4. Merge Sort
5. Quick Sort

**Code(Using Selection Sort):**

|  |
| --- |
| #include<stdio.h> #include<time.h>  void selection\_sort(int[],int);  void main() {  int a[10000],n,i;  time\_t;  double time\_taken;  clock\_t start,end;  printf("21012021003\_AMIT GOSWAMI\n"); printf("Enter number of elements: "); scanf("%d",&n);  srand((unsigned) time(&t)); for(i=0;i<n;i++)  {  a[i]=rand() % 1000;  } start=clock();  selection\_sort(a,n);  end=clock();  printf("\nSorted randomly generated elements are: "); |

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

{

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

}

time\_taken=(double)(end-start)/CLOCKS\_PER\_SEC;

printf("\nTime taken by selection sort for random unsorted array: %lf",time\_taken);

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

a[i]=i+1;

}

start=clock();

selection\_sort(a,n);

end=clock();

printf("\n\nSorted elements are: ");

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

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

}

time\_taken=(double)(end-start)/CLOCKS\_PER\_SEC;

printf("\nTime taken by selection sort for sorted array%lf",time\_taken);

}

void selection\_sort(int a[],int n)

{

int i,j,min,temp;

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

{

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

{ if(a[min]>a[j])

{ min=j;

}

}

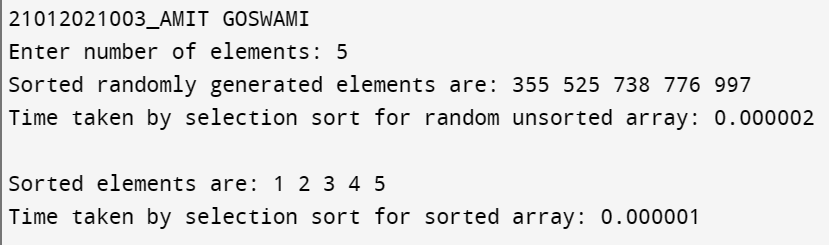
if(min!=i) { temp=a[i]; a[i]=a[min]; a[min]=temp;

}

}

}

**Output:**



**Analysis:**

|  |  |  |
| --- | --- | --- |
| Input Size | Sorted | Random |
| 5 | 0.000001 | 0.000002 |
| 25 | 0.000003 | 0.000005 |
| 500 | 0.000425 | 0.000433 |
| 5000 | 0.033747 | 0.034069 |
| 10000 | 0.152171 | 0.162309 |

0

0.05

0.1

0.15

0.2

0

2000

4000

6000

8000

10000

12000

Time Taken

Input Size

Selection Sort

Sorted

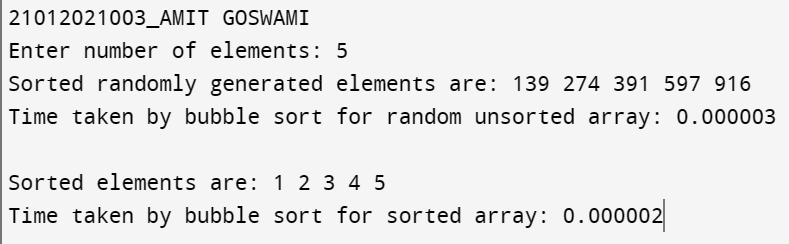
Random

**Code(Using Bubble Sort):**

|  |
| --- |
| #include<stdio.h> #include<time.h>  void bubble\_sort(int [],int); void main()  {  int a[10000],n,i; time\_t t;  double time\_taken; clock\_t start,end;  printf("21012021003\_AMIT GOSWAMI \n"); printf("Enter number of elements: "); scanf("%d",&n); srand((unsigned) time(&t)); for(i=0;i<n;i++)  {  a[i]=rand() %1000;  }  start=clock(); bubble\_sort(a,n);  end=clock();  printf("\nSorted randomly generated elements are: "); for(i=0;i<n;i++)  {  printf("%d ",a[i]);  }  time\_taken=(double)(end-start)/CLOCKS\_PER\_SEC;  printf("\nTime taken by bubble sort for random unsorted array: %lf",time\_taken); for(i=0;i<n;i++)  { a[i]=i+1;  } |

|  |
| --- |
| start=clock();  bubble\_sort(a,n);  end=clock();  printf("\n\nSorted elements are:"); for(i=0;i<n;i++)  {  printf("%d ",a[i]);  }  time\_taken=(double)(end-start)/CLOCKS\_PER\_SEC;   printf("\nTime taken by bubble sort for sorted array: %lf",time\_taken);  }  void bubble\_sort(int a[],int n)  {  int i,j,temp,flag; for(i=0;i<n-1;i++)  {  flag=0; for(j=0;j<n-1-i;j++)  { if(a[j]>a[j+1])  {  temp=a[j]; a[j]=a[j+1]; a[j+1]=temp; flag=1;  }  } if(flag==0)  { break;  }  }  } |

**Output:**



**Analysis:**

|  |  |  |
| --- | --- | --- |
| Input Size | Sorted | Random |
| 5 | 0.000002 | 0.000003 |
| 25 | 0.000002 | 0.000005 |
| 500 | 0.000005 | 0.000581 |
| 5000 | 0.000016 | 0.055213 |
| 10000 | 0.000035 | 0.208471 |

0

0.05

0.1

0.15

0.2

0.25

0

2000

4000

6000

8000

10000

12000

Time Taken

Input Size

Bubble Sort

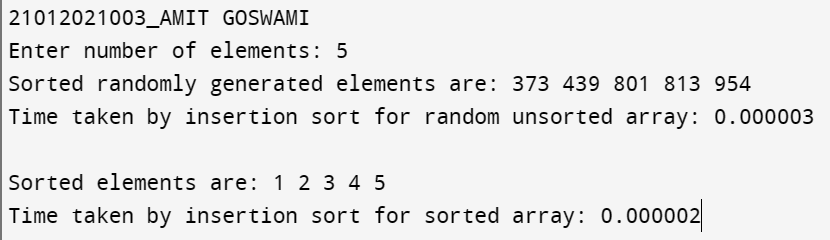
Sorted

Random

**Code(Using Insertion Sort):**

|  |
| --- |
| #include<stdio.h> #include<time.h>  void insertion\_sort(int [],int); void main()  {  int a[10000],n,i; time\_t t;  double time\_taken; clock\_t start,end;  printf("21012021003\_AMIT GOSWAMI \n"); printf("Enter number of elements: "); scanf("%d",&n);  srand((unsigned) time(&t));  for(i=0;i<n;i++)  {  a[i]=rand() % 1000;  }  start=clock();  insertion\_sort(a,n);  end=clock();  printf("\nSorted randomly generated elements are: "); for(i=0;i<n;i++)  {  printf("%d ",a[i]);  }  time\_taken=(double)(end-start)/CLOCKS\_PER\_SEC;  printf("\nTime taken by insertion sort for random unsorted array: %lf",time\_taken); for(i=0;i<n;i++)  { a[i]=i+1;  } |
| start=clock();  insertion\_sort(a,n);  end=clock();  printf("\n\nSorted elements are: "); for(i=0;i<n;i++)  {  printf("%d ",a[i]);  }  time\_taken=(double)(end-start)/CLOCKS\_PER\_SEC;  printf("\nTime taken by insertion sort for sorted array: %lf",time\_taken);  }  void insertion\_sort(int a[],int n)  {  int i,j,temp; for(i=1;i<n;i++)  { temp=a[i]; j=i-1;  while(j>=0 && a[j]>temp)  { a[j+1]=a[j];  j--;  }  a[j+1]=temp;  }  } |

**Output:**



**Analysis:**

|  |  |  |
| --- | --- | --- |
| Input Size | Sorted | Random |
| 5 | 0.000002 | 0.000003 |
| 25 | 0.000001 | 0.000003 |
| 500 | 0.000004 | 0.000205 |
| 5000 | 0.000034 | 0.029683 |
| 10000 | 0.000067 | 0.091895 |

0

0.02

0.04

0.06

0.08

0.1

0

2000

4000

6000

8000

10000

12000

Time Taken

Input Size

Insertion sort

Sorted

Random

**Code(Using Merge Sort):**

|  |
| --- |
| #include<stdio.h>  #include<time.h>  int a[10000],b[10000];  void merge(int a[],int lb,int mid,int ub)  {  int i,j,k; i=lb; j=mid+1; k=lb;  while(i<=mid && j<=ub)  {  if(a[i]<=a[j])  { b[k]=a[i]; i++;  k++;  }  else{ b[k]=a[j]; j++;  k++;  }  }  if(i>mid)  { while(j<=ub)  {  b[k]=a[j]; j++;  k++;  }  } else{ |

while(i<=mid) {

b[k]=a[i]; i++;

k++;

} } for(i=lb;i<=ub;i++)

{ a[i]=b[i];

}

}

void merge\_sort(int a[],int lb,int ub) {

int mid;

if(lb<ub) {

mid=(lb+ub)/2; merge\_sort(a,lb,mid); merge\_sort(a,mid+1,ub); merge(a,lb,mid,ub);

}

}

void main() {

int n,i,j;

time\_t t;

double time\_taken; clock\_t start,end;

printf("21012021003\_AMIT GOSWAMI \n"); printf("Enter number of elements: "); scanf("%d",&n);

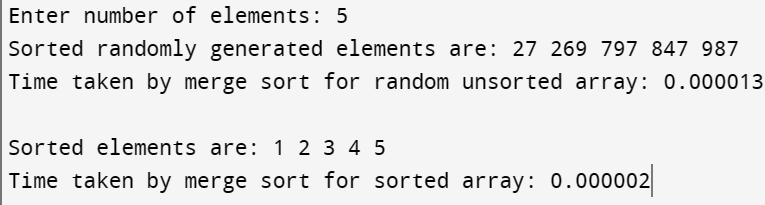
srand((unsigned) time(&t)); for(i=0;i<n;i++) { a[i]=rand() % 1000;

}

start=clock(); merge\_sort(a,0,n-1); end=clock();

|  |
| --- |
| printf("\nSorted randomly generated elements are: "); for(i=0;i<n;i++)  {  printf("%d",a[i]);  }  time\_taken=(double)(end-start)/CLOCKS\_PER\_SEC;  printf("\nTime taken by merge sort for random unsorted array:%lf",time\_taken); for(i=0;i<n;i++)  { a[i]=i+1;  } start=clock();  merge\_sort(a,0,n-1);  end=clock();  printf("\n\nSorted elements are:"); for(i=0;i<n;i++)  { printf("%d",a[i]);  }  time\_taken=(double)(end-start)/CLOCKS\_PER\_SEC;  printf("\nTime taken by merge sort for sorted array%lf",time\_taken);  } |

**Output:**



**Analysis:**

|  |  |  |
| --- | --- | --- |
| Input Size | Sorted | Random |
| 5 | 0.000002 | 0.000013 |
| 25 | 0.000002 | 0.000016 |
| 500 | 0.000025 | 0.000061 |
| 5000 | 0.000342 | 0.001001 |
| 10000 | 0.000888 | 0.001337 |

0

0.0002

0.0004

0.0006

0.0008

0.001

0.0012

0.0014

0.0016

0

2000

4000

6000

8000

10000

12000

Time Taken

Input Size

Merge Sort

Sorted

Random

**Code(Using Quick Sort):**

|  |
| --- |
| #include<stdio.h> #include<time.h>  void quick\_sort(int a[],int lb,int ub)  {  int pivot,start,end,temp; if(lb<ub)  {  pivot=lb; start=lb; end=ub; while(start<end)  { while(a[pivot]>=a[start] )  { start++;  } while(a[pivot]<a[end])  {  end--;  } if(start<end)  {  temp=a[start]; a[start]=a[end]; a[end]=temp;  }  }  temp=a[pivot]; a[pivot]=a[end]; a[end]=temp;    quick\_sort(a,lb,end-1); |

quick\_sort(a,end+1,ub);

}

}

int main() {

int a[10000],i,n; time\_t t;

double time\_taken; clock\_t start,end;

printf("21012021003\_AMIT GOSWAMI \n"); printf("Enter number of elements: "); scanf("%d",&n);

srand((unsigned) time(&t)); for(i=0;i<n;i++)

{

a[i]=rand()%1000;

}

start=clock();

quick\_sort(a,0,n-1);

end=clock();

printf("\nSorted randomly generated elements are");

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

{

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

}

time\_taken=(double)(end-start)/CLOCKS\_PER\_SEC;

printf("\nTime taken by quick sort for random unsorted array: %lf",time\_taken); for(i=0;i<n;i++)

{

a[i]=i+1;

}

start=clock();

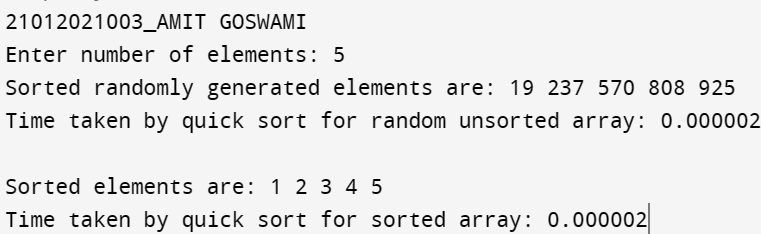
quick\_sort(a,0,n-1);

end=clock();

printf("\n\nSorted elements are: ");

|  |
| --- |
| for(i=0;i<n;i++)  {  printf("%d ",a[i]);  }  time\_taken=(double)(end-start)/CLOCKS\_PER\_SEC; printf("\nTime taken by quick sort for sorted array: %lf",time\_taken); return 0;  } |

**Output:**



**Analysis:**

|  |  |  |
| --- | --- | --- |
| Input Size | Sorted | Random |
| 5 | 0.000002 | 0.00002 |
| 25 | 0.000003 | 0.000005 |
| 500 | 0.000397 | 0.000107 |
| 5000 | 0.032309 | 0.000820 |
| 10000 | 0.118116 | 0.001367 |

0

0.02

0.04

0.06

0.08

0.1

0.12

0.14

0

2000

4000

6000

8000

10000

12000

Time Taken

Input Size

Quick Sort

Sorted

Random

**Analysis of all sorting types (Sorted Data):**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input Size | Selection Sort | Bubble Sort | Insertion Sort | Merge Sort | Quick Sort |
| 5 | 0.000002 | 0.000001 | 0.000002 | 0.000001 | 0.000002 |
| 25 | 0.000003 | 0.000002 | 0.000001 | 0.000002 | 0.000003 |
| 500 | 0.000425 | 0.000005 | 0.000004 | 0.000025 | 0.000397 |
| 5000 | 0.033747 | 0.000016 | 0.000034 | 0.000342 | 0.032309 |
| 10000 | 0.152171 | 0.000035 | 0.000067 | 0.000888 | 0.118116 |

0

0.02

0.04

0.06

0.08

0.1

0.12

0.14

0.16

5

25

500

5000

10000

Time Taken

Input Size

All Sorting Algorithms (Sorted Data)

Selection

Bubble

Insertion

Merge

Quick

**Analysis of all sorting types (Random Data):**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input Size | Selection Sort | Bubble Sort | Insertion Sort | Merge Sort | Quick Sort |
| 5 | 0.000002 | 0.000003 | 0.000003 | 0.000013 | 0.00002 |
| 25 | 0.000005 | 0.000005 | 0.000003 | 0.000016 | 0.000005 |
| 500 | 0.000433 | 0.000581 | 0.000205 | 0.000061 | 0.000107 |
| 5000 | 0.034069 | 0.055213 | 0.029683 | 0.001001 | 0.000820 |
| 10000 | 0.162309 | 0.208471 | 0.091895 | 0.001337 | 0.001367 |

0

0.05

0.1

0.15

0.2

0.25

5

25

500

5000

10000

Time Taken

Input Size

All Sorting Algorithms (Random Data)

Selection

Bubble

Insertion

Merge

Quick

**Conclusion:**

Various sorting algorithms perform differently for sorted and unsorted data. Thus it is advisable to choose correct algorithms according to user’s needs.