SORTING

Sorting is the technique of arranging the elements in a given order. It may be i. ascending (in increasing order) ii. Descending (decreasing order)

There are many techniques to sort elements out of those the common techniques are-

1. Bubble Sorting

2. Insertion Sorting

3. Selection Sorting

4. Merge Sorting

5. Radix Sorting

6. Heap Sorting

7. Quick Sorting

1. **Bubble Sorting** (Refer bsort.c)

Bubble sorting is something like pair sorting. Two consecutive elements are bubbled (or paired) for comparison. Similar procedure is followed for n-1 number of passes (i.e. rounds) where n is the number of elements in the array. In each pass there would be n-i comparisons where i is the pass value (or round number).

For example, if there are 7 elements then we must process 6 (n-1) passes. In first pass there would be 6 comparisons, in 2nd pass there would be 5 comparisons and so on.

**Explanation:**

**Let the array of 7 elements is, our attempt is to sort the array ascending in order**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 5 | 2 | 9 | 7 | 3 | 4 | 6 |

Pass1:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 5 | 2 | 9 | 7 | 3 | 4 | 6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 9 | 7 | 3 | 4 | 6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 9 | 7 | 3 | 4 | 6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 7 | 9 | 3 | 4 | 6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 7 | 3 | 9 | 4 | 6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 7 | 3 | 4 | 9 | 6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 7 | 3 | 4 | 6 | 9 |

Pass 2:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 7 | 3 | 4 | 6 | 9 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 7 | 3 | 4 | 6 | 9 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 7 | 3 | 4 | 6 | 9 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 3 | 7 | 4 | 6 | 9 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 3 | 4 | 7 | 6 | 9 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 3 | 4 | 6 | 7 | 9 |

And there would be similar processing in other passes finally to get the sorted array.

**C code:**

void bsort(int \*a, int n)

{

int i,j,t;

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

{

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

{

if(a[j]>a[j+1])

{

t=a[j];

a[j]=a[j+1];

a[j+1]=t;

}

}

}

}

**Algorithm:**

If ARR be the set of data of N elements

Let I and J be the counter variables

TEMP is the variable for temporary purpose.

1. Start

2. Initialize counter I to 1

I :=1

3. Repeat Step #4 to #6 until I is less then N

4. Initialize J to 0

J := 0

5. Repeat step #6 while J is less than N-I

6. If ARR[J] > ARR[J+1]

Swap

TEMP := ARR[J]

ARR[J] := ARR[J+1]

ARR[J+1] := TEMP

End of if block

End of inner loop

End of outer loop

7. Stop.

2. Insertion Sorting (Refer isort.c)

Insertion sort as its name-shake, an element is picked and inserted at its appropriate position to get final sorted array.

**Explanation:**

**Let the array of 7 elements is, our attempt is to sort the array ascending in order**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 5 | 2 | 9 | 7 | 3 | 4 | 6 |

Here skip the first element and start verifying the appropriate position of the elements starting from position number 2.

Pass1:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 5 | 2 | 9 | 7 | 3 | 4 | 6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 9 | 7 | 3 | 4 | 6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 9 | 7 | 3 | 4 | 6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 7 | 9 | 3 | 4 | 6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 3 | 5 | 7 | 9 | 4 | 6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 3 | 4 | 5 | 7 | 9 | 6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 3 | 4 | 5 | 6 | 7 | 9 |

**C Code:**

void isort(int \*a, int n)

{

int i,j,t;

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

{

for(j=i;j>0;j--)

{

if(a[j]<a[j-1])

{

t=a[j];

a[j]=a[j-1];

a[j-1]=t;

}

}

}

}

**Algorithm**

If ARR be the set of data of N elements

Let I and J be the counter variables

TEMP is the variable for temporary purpose.

1. Start

2. Initialise the counter I with 1 [index of 2nd element]

I := 1

3. Repeat step #4 to 6 for I<N

4. Initialise J with I

J := I

5. Repeat step 6 until J > 0

6. If ARR[J] < ARR [J-1] then

Swap

TEMP := ARR[J]

ARR[J] := ARR[J-1]

ARR[J-1] := TEMP

End of if block

End of inner loop (with step 4)

End of Outer loop (with setp 2)

7. Stop

Selection Sort (Refer ssort.c)

It is something reverse processing to the insertion sorting. In the former case we have to pick an element and then it must be inserted at its right position but in selection sort, the elementary position is decided and the right element is brought to that position.

**Explanation:**

**Let the array of 7 elements is, our attempt is to sort the array ascending in order**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 5 | 2 | 9 | 7 | 3 | 4 | 6 |

Pass 1:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 5 | 2 | 9 | 7 | 3 | 4 | 6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 9 | 7 | 3 | 4 | 6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 9 | 7 | 3 | 4 | 6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 9 | 7 | 3 | 4 | 6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 9 | 7 | 3 | 4 | 6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 9 | 7 | 3 | 4 | 6 |

Pass 2:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 9 | 7 | 3 | 4 | 6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 9 | 7 | 3 | 4 | 6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 9 | 7 | 3 | 4 | 6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 3 | 9 | 7 | 5 | 4 | 6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 3 | 9 | 7 | 5 | 4 | 6 |

**C Code**

void ssort(int \*a, int n)

{

int i,j,k,t;

int min;

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

{

min=a[i];

k=i;

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

{

if(min>a[j])

{

min=a[j];

k=j;

}

}

t=a[i];

a[i]=a[k];

a[k]=t;

}

}

**Algorithm**

If ARR be the set of data of N elements

Let I and J be the counter variables

TEMP is the variable for temporary purpose.

Two additional variables required to be maintained are

MIN to contain the minimum value and K is the position

of the minimum value

1. Start.

2. Initialise I with 0

I := 0

3. Repeat step #4 to 6 until I less then N-1

4. Store first element in MIN

MIN := ARR[I]

5. Identify the smallest element

a. Initialise J to I+1 and K to I

J := I + 1

K := I

b. Repeat until J < N

if MIN > ARR[J]

MIN := ARR[J]

K := J

End of if block

End of repeatation block

6. Swap data at minmum position with element at required pos

TEMP := ARR[I]

ARR[I := ARR[K]

ARR[K] := TEMP

6. Stop.

4. Merge Sort (refer msort.c, msort1.c)

**C Code**

void msort(int \*a, int \*b, int \*c, int m, int n)

{

int i=0,j=0,k;

for(k=0;k<(m+n);k++)

{

if( a[i] > b[j] ) && (j<n) )

{

c[k] = b[j];

j++;

}

else

{

c[k] = a[i];

i++;

}

}

}

**Algorithm**

Let A and B are two sorted array, they are required to be

merger in sorted way.

Let C is the array to contain the result.

Maintain three counter variables I, J and K for 3 different

array.

1. Start

2. Initialise all the counters to 0

I := 0

J := 0

K := 0

3. Repeat until K is less than the sum of both the

source array (say M and N )

a. Check element of A array at I position with

element of B array at J position and check whether

the B array is not empty

if A[I] > B[J] and J < N

C[K] := B[j]

increment J

J := J + 1

Otherwise

If I < M

C[K] := A[I]

increment I

I := I + 1

Otherwise

C[K] := B[J]

J := J+1

End if

End if

b. Increment K

K := K + 1

End of prepeatation

4. Stop

5. Radix Sort (refer rsort.c)

**Explanation:**

Radix sort is based on the digits in each element. So all the elements must be converted to uniform set of digits before they are passed for sorting.

Example:

if the set of elements are

134, 67, 85, 239, 7, 392, 494

They must be converted into

134 067, 085, 239, 007, 392, 494

Note: by inserting a precedence zero to the

Number does not alter its value.

According to the number of digits the elements must be processed. i.e. if there are 3 digit elements we must sort them in 3 phase.

For each phase we must define 10 pockets (or sets) number from 0 to 9 to store matched element.

Let the original array of 6 elements is

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 143 | 67 | 285 | 239 | 7 | 392 | 494 |

**Explanation**

Pass 1: Here the digit at 1th place is considered

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Original Data | Pocket 0 | Pocket 1 | Pocket 2 | Pocket 3 | Pocket 4 | Pocket 5 | Pocket 6 | Pocket 7 | Pocket 8 | Pocket 9 |
| 143 |  |  |  | 143 |  |  |  |  |  |  |
| 067 |  |  |  |  |  |  |  | 067 |  |  |
| 285 |  |  |  |  |  | 285 |  |  |  |  |
| 239 |  |  |  |  |  |  |  |  |  | 239 |
| 007 |  |  |  |  |  |  |  | 007 |  |  |
| 392 |  |  | 392 |  |  |  |  |  |  |  |
| 494 |  |  |  |  | 494 |  |  |  |  |  |

Now the elements are collected together from pocket wise top –down manner

So the collection would be

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 392 | 143 | 494 | 285 | 067 | 007 | 239 |

Now reset all in the pockets in the next pass as it is done the previous however this time take the digit at 10th place

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Original Numbers | Pocket 0 | Pocket 1 | Pocket 2 | Pocket 3 | Pocket 4 | Pocket 5 | Pocket 6 | Pocket 7 | Pocket 8 | Pocket 9 |
| 392 |  |  |  |  |  |  |  |  |  | 392 |
| 143 |  |  |  |  | 143 |  |  |  |  |  |
| 494 |  |  |  |  |  |  |  |  |  | 494 |
| 285 |  |  |  |  |  |  |  |  | 285 |  |
| 067 |  |  |  |  |  |  | 067 |  |  |  |
| 007 | 007 |  |  |  |  |  |  |  |  |  |
| 239 |  |  |  | 239 |  |  |  |  |  |  |

Now the elements are collected together from pocket wise top –down manner

So the collection would be

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 007 | 239 | 143 | 067 | 285 | 392 | 494 |

Pass 3:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Original Numbers | Pocket 0 | Pocket 1 | Pocket 2 | Pocket 3 | Pocket 4 | Pocket 5 | Pocket 6 | Pocket 7 | Pocket 8 | Pocket 9 |
| 007 | 007 |  |  |  |  |  |  |  |  |  |
| 239 |  |  | 239 |  |  |  |  |  |  |  |
| 143 |  | 143 |  |  |  |  |  |  |  |  |
| 067 | 067 |  |  |  |  |  |  |  |  |  |
| 285 |  |  | 285 |  |  |  |  |  |  |  |
| 392 |  |  |  | 392 |  |  |  |  |  |  |
| 494 |  |  |  |  | 494 |  |  |  |  |  |

Now the elements are collected together from pocket wise top –down manner

So the collection would be

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 007 | 067 | 143 | 239 | 285 | 392 | 494 |

**Algorithm**

Let ARR be the unsorted array

Take CNT a counter variable for number of passes

let DIG be the value of a digit of a number

Let POK[10] are 10 pockets to contain elements

1.Start

2. Convert all elements to their uniform digit set

3. Get Total number of digits per elements

Say it is N

4. Initialise CNT to 1

CNT := 1

5. Repeat until CNT < = N

a. Get the digit nth position (say it is 1th, 10th,

100th etc) and pass it to appropriate pocket

POK[Digit] = Element

b. Incremen the pocket counter

End of repeatation

c. Set all elements to the origional array pocket-wise

from top to down.

d. Increment counter

CNT := CNT + 1

End of repeatation

4. Stop.

6. Heap Sort (refer rsort.c)

**C Code**

# include<stdio.h>

void heap\_sort(int \*, int );

void create\_heap(int \*, int);

void display(int \*, int);

/\* Definition of the function \*/

void create\_heap(int list[], int n )

{

int k, j, i, temp;

for(k = 2 ; k <= n; ++k)

{

i = k ;

temp = list[k];

j = i / 2 ;

while((i > 1) && (temp > list[j]))

{

list[i] = list[j];

i = j ;

j = i / 2 ;

if ( j < 1 )

j = 1 ;

}

list[i] = temp ;

}

}

/\* End of heap creation function \*/

/\* Definition of the function \*/

void heap\_sort(int list[], int n)

{

int k, temp, value, j, i, p;

int step = 1;

for(k = n ; k >= 2; --k)

{

temp = list[1] ;

list[1] = list[k];

list[k] = temp ;

i = 1 ;

value = list[1];

j = 2 ;

if((j+1) < k)

if(list[j+1] > list[j])

j ++;

while((j <= ( k-1)) && (list[j] > value))

{

list[i] = list[j];

i = j ;

j = 2\*i ;

if((j+1) < k)

if(list[j+1] > list[j])

j++;

else

if( j > n)

j = n ;

list[i] = value;

} /\* end of while statement \*/

printf("\n Step = %d ", step);

step++;

for(p = 1; p <= n; p++)

printf(" %d", list[p]);

} /\* end for loop \*/

}

/\* Display function \*/

void display(int list[], int n)

{

int i;

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

{

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

}

}

/\* Function main \*/

void main()

{

int list[100];

int i, size = 13 ;

printf("\n Size of the list: %d", size);

for(i = 1 ; i <= size ; ++i)

{

list[i] = rand() % 100;

}

printf("\n Entered list is as follows:\n");

display(list, size);

create\_heap(list, size);

printf("\n Heap\n");

display(list, size);

printf("\n\n");

heap\_sort(list,size);

printf("\n\n Sorted list is as follows :\n\n");

display(list,size);

}

NOTE: The above heap sort program includes main() too.

7. Quick Sort (refer qsort.c)

**C Code**

#include<stdio.h>

#include<conio.h>

void q\_sort(int a[],int l, int h);

void main()

{

int x[10]={2,6,3,9,1,5,4,2,8,4};

int i;

int l=0,h=9;

q\_sort(x,l,h);

for(i=0;i<10;i++)

{

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

}

getch();

}

void q\_sort(int x[], int l, int h)

{

int t;

int k,low,high;

clrscr();

low=l;

high=h;

k=x[(low+high)/2]; // Copy of the middle element

do

{

while(k>x[low])

{

low++;

}

while(k<x[high])

{

high--;

}

if(low<=high)

{

t=x[low];

x[low++]=x[high];

x[high--]=t;

}

}while(low<=high);

if(l<high)

q\_sort(x,l,high);

if(low<h)

q\_sort(x,low,h);

}

NOTE: The above quick sort program includes main() too.

Topological sorting

Let G be a directed graph without circle (or a partially ordered set –POSET), a topological sort of S is a linear ordering of nodes in the graph which preserves the partial ordering.

That is***, if*** ***u*** ***v (i.e. there is a path from u to v) in the graph, then u comes before v in the linear ordering.***

The above graph has no cycle and the vertex are in partial ordered set (POSET), to find out the topological sorting of the above graph following algorithm should be used.

Algorithm

Let G be the graph of n number of vertices without circle.

Maintain a Queue as intermediate data structure to keep sorted nodes. Let FRONT be the front end iof the queue that follows FIFO.

1. Start
2. Find out the **in-degree** INDEG(N) of all the nodes in the graph.
3. Keep the all nodes that have zero in-degree into the queue.
4. Repeat until the queue is empty
   1. Remove the front node and set

FRONT:=FRONT+1

* 1. Repeat until there is no neighbor (adjacent node) M of the processed node
     1. Decrement the in-degree of the neighbor

INDEG(M) :=INDEG-1

* + 1. If INDEG(M)=0 then add the node to the queue

End of repetition

End of repetition

1. stop

After doing the above algorithm the linear collection of the nodes would be as given diagram that also preserves the previous partial ordering as directed through arrow marks.

**NOTE:** For solution you must draw the adjacent list and then you have to get the in-degree of each node. The node that has zero in degree are said to be source. So, these are first put into the queue. Further their neighbors with lowest in-degree and so on.