1. **Activity: Sort List oi elements stored in a linked list using buuble sort algorithm**

**Algorithm:**

1. Swap\_count = 1
2. While swap\_count != 0:
   1. Swap\_count = 0
   2. If(arr[i] > arr[i+1])
      1. Swap arr[i] and arr[i+1]
      2. Swap\_count++
   3. i++

**Test Case Table:**

|  |  |  |
| --- | --- | --- |
| Input | Sorted | Pass/Fail |
| 33, 20, 4, 3, 7, 4, 2, 4 | 2, 3, 4, 4, 4, 7, 20, 33 | Pass |
| 3, 7, 4, 2, 4 | 2, 3, 4, 4, 7 | Pass |

**Program:**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

struct Node{

    int data;

    struct Node \*next;

} \*head = NULL, \*ptr = NULL;

void insertbeg(int n){

    struct Node \*temp = NULL;

    temp = (struct Node \*)malloc(sizeof(struct Node));

    if (temp == NULL)

    {

        printf("Memory full, could not add.\n");

        return;

    }

    temp->data = n;

    if (head == NULL)

    {

        temp->next = NULL;

    }

    else{temp->next = head;}

    head = temp;

}

void display(){

    printf("List: ");

    for(ptr = head; ptr != NULL; ptr = ptr->next){

        printf("%d --> ", ptr->data);

    }

    printf("END\n");

}

void deletebeg(){

    struct Node \*temp = head;

    head = temp->next;

    free(temp);

    printf("Deleteion Successful\n");

}

int list\_size(){

    int size = 0;

    for(ptr = head; ptr != NULL; ptr = ptr->next) size++;

    return size;

}

//(ws) enables with steps.

// ws = 0 (no step), ws != 0 (steps will be shown)

void bubble\_sort(int ws){

    if (head == NULL)

    {

        printf("Empty list");

        return;

    }

    int i, j, n = list\_size();

    int swap = 1;

    while (swap){

        for(i = 0; i < n-1; i++){

            swap = 0;

            for(j = 0, ptr=head; j < (n-1)-i; j++, ptr = ptr->next){

                if (ptr->data > ptr->next->data)

                {

                    int t = ptr->data;

                    ptr->data = ptr->next->data;

                    ptr->next->data = t;

                    swap++;

                }

                }

            if (swap == 0)

            {

                printf("\nSorted -->\n");

                display();

                break;

            }

            else {

                if (ws){

                    printf("Pass %d ->\n", i+1);

                    display();

                }

            }

        }

    }

}

int main(){

    insertbeg(4);

    insertbeg(2);

    insertbeg(4);

    insertbeg(7);

    insertbeg(3);

    printf("Sanmple data inserted..\n");

    display();

    int check = 1;

    char choice;

    while(check){

        printf("Do you want to insert data (y/n): ");

        scanf("%c", &choice);

        switch (tolower(choice))

        {

        case 'y':

            printf("Enter number you want to insert: ");

            int n;

            scanf("%d", &n);

            insertbeg(n);

            break;

        case 'n':

            check = 0;

            break;

        default:

            printf("Invalid selection.\n");

            break;

        }

        while((getchar()) != '\n');

    }

    display();

    printf("\n");

    check = 1;

    int c;

    while(check){

        printf("Choose sort type\n");

        printf("1. Sort with steps\n");

        printf("2. Sort without steps\n");

        printf("Enter selection (1/2): ");

        scanf("%d", &c);

        switch (c)

        {

        case 1:

            bubble\_sort(1);

            check = 0;

            break;

        case 2:

            bubble\_sort(0);

            check = 0;

            break;

        default:

            printf("Invalid selection.\n");

            break;

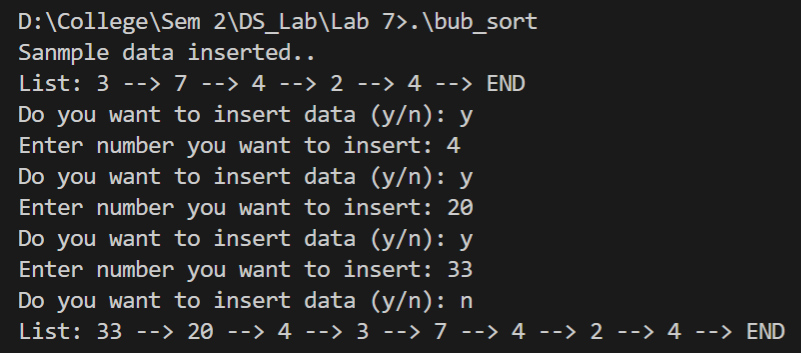
        }

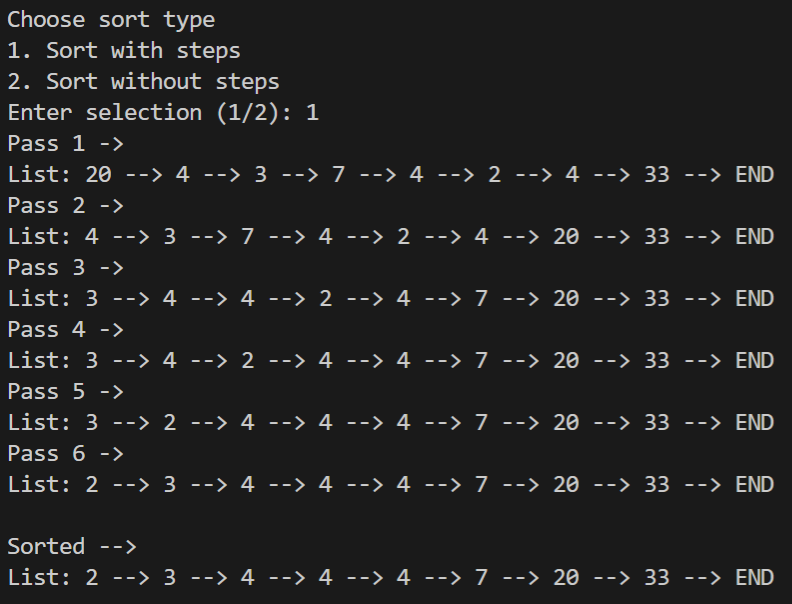
    }

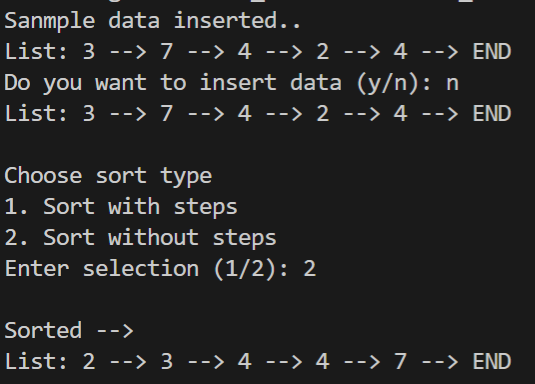
    return 0;

}

**Screenshot of compilation and execution:**

****

****

****

1. **Activity: Sort list of Elements stored in an array using: Insertion sort, Selection Sort, Merge Sort, Quick Sort.**

**Algorithm:**

1. Insertion sort:
   1. For i=0 to n-1
      1. Element = arr[i]
      2. J=i
      3. While(j>0 & arr[j-1]>element)
         1. Arr[j] = arr[j-1]
         2. J—
      4. Arr[j] = element
2. Selection Sort:
   1. For i=0 to n-2
      1. Min = i
      2. For j=i+1 to n-1
         1. If(arr[j] < arr[min])
            1. Min = j
      3. If min != i
         1. Swap arr[min] and arr[i]
3. Merge Sort
   1. Merge function(arr, left, mid, right)
      1. Len1 = mid – left + 1
      2. Len2 = right – mid
      3. Make 2 arrarys: lArr[len1] & rArr[len2]
      4. For i=0 to len1 -1
         1. lArr[i] = arr[left+i]
      5. For I=0 to len2 -1
         1. rArr[i] = arr[mid+i]
      6. i=0, j=0, k=left
      7. while(i<len1 & j<len2)
         1. if(lArr[i] <= rArr[j])
            1. arr[k] = lArr[i]
            2. i++
         2. else
            1. arr[k] = rArr[j]
            2. j++
         3. k++
      8. while(i<len1)
         1. arr[k] = lArr[i]
         2. i++
         3. k++
      9. while(j<len2)
         1. arr[k] = rArr[j]
         2. j++
         3. k++
   2. MergeSort function(arr, left, right)
      1. If left >= right, return
      2. Mid = (left+right)/2
      3. MergeSort(arr, left, mid)
      4. MergeSort(arr, mid+1, right)
      5. Merge(arr, left, mid, right)
4. Quick Sort
   1. Partition function(arr, start, end)
      1. Pivot = arr[end]
      2. Wall = (start -1)
      3. For i=start to end-1
         1. If arr[i] < pivot
            1. Wall++
            2. Swap arr[wall] & arr[i]
      4. Swap arr[wall+1] & arr[end]
      5. Return (wall+1)
   2. QuickSort function(arr, start, end)
      1. if (start < end)
         1. part\_index = Partition(arr, start, end)
         2. QuickSort(arr, start, part\_index -1)
         3. Quicksort(arr, part\_index + 1, end)

**Program:**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <ctype.h>

//Determines size of generated array

int size = 5;

//Print array function

//Also can wall between sorted and unsroted parts

//No wall parameter is any number out of arrays index range

void printArr(int arr[], int wall){

    for (int i = 0; i < size; i++)

    {

        printf("%d ", arr[i]);

        if (i == wall)

        {

            printf("| ");

        }

    }

    printf("\n");

}

void insertion\_sort(int arr[], int visualise){

    for (int i = 0; i < size; i++)

    {

        int element = arr[i];

        int j = i;

        while (j>0 && arr[j-1] > element)

        {

            arr[j] = arr[j-1];

            j--;

        }

        arr[j] = element;

        if(visualise) printf("Pass %d->\n", i+1);

        if(visualise || i == size-1) printArr(arr, i);

    }

}

void selection\_sort(int arr[], int visual){

    int i;

    for (i = 0; i < size-1; i++)

    {

        int min = i;

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

        {

            if (arr[j] < arr[min])

            {

                min = j;

            }

        }

        if (min != i)

        {

            int temp = arr[min];

            arr[min] = arr[i];

            arr[i] = temp;

        }

        if(visual) printf("Pass %d (Min term: %d)->\n", i+1, arr[i]);

        if(visual) printArr(arr, i);

    }

    if(visual) printf("Pass %d (Min term: %d)->\n", i+1, arr[i]);

    printArr(arr, i);

}

void merge(int arr[], int left, int mid, int right, int visual){

    int len1 = mid-left+1;

    int len2 = right-mid;

    int ar1[len1], ar2[len2];

    for (int i = 0; i < len1; i++)

    {

        ar1[i] = arr[left+i];

    }

    for (int i = 0; i < len2; i++)

    {

        ar2[i] = arr[mid + i + 1];

    }

    int i=0,j=0,k=left;

    while (i < len1 && j < len2)

    {

        if (ar1[i] <= ar2[j])

        {

            arr[k] = ar1[i];

            i++;

        }

        else{

            arr[k] = ar2[j];

            j++;

        }

        k++;

    }

    while (i < len1)

    {

        arr[k] = ar1[i];

        i++;

        k++;

    }

    while (j < len2)

    {

        arr[k] = ar2[j];

        j++;

        k++;

    }

    if (visual) printf("Merge( ");

    if (visual){

        printf("Array 1: ");

        for (int a = 0; a < i; a++)

        {

            printf("%d ", ar1[a]);

        }

        printf(" ");

        printf("Array 2: ");

        for (int a = 0; a < j; a++)

        {

            printf("%d ", ar2[a]);

        }

        printf(")->\nMerged array: ");

        for (int a = left; a < k; a++)

        {

            printf("%d ", arr[a]);

        }

    }

}

void mergeSort(int arr[], int left, int right, int visual){

    if (left < right) {

        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid, visual);

        mergeSort(arr, mid + 1, right, visual);

        if(visual) printf("\n");

        merge(arr, left, mid, right, visual);

    }

}

int partition(int arr[], int start, int end, int visual){

    int pivot = arr[end];

    int wall = (start - 1);

    for (int i = start; i <= end-1; i++)

    {

        if (arr[i] <= pivot)

        {

            wall++;

            int temp = arr[wall];

            arr[wall] = arr[i];

            arr[i] = temp;

        }

    }

    int temp = arr[wall + 1];

    arr[wall + 1] = arr[end];

    arr[end] = temp;

    if(visual){printf("Partition: %d\n", arr[wall+1]);

    printArr(arr, (wall+1));}

    return (wall+1);

}

void Quicksort(int arr[], int start, int end, int visual){

    if(start < end){

        int part\_index = partition(arr, start, end, visual);

        Quicksort(arr, start, part\_index - 1, visual);

        Quicksort(arr, part\_index+1, end, visual);

    }

}

int main(){

    int visual = 0;

    //Creates random equivalent arrays

    srand(time(0));

    int arr[size], insert\_sort[size], select\_sort[size], merge\_sort[size], quick\_sort[size];

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

    {

        int element = (rand() % (size+10));

        arr[i] = element;

        insert\_sort[i] = element;

        select\_sort[i] = element;

        merge\_sort[i] = element;

        quick\_sort[i] = element;

    }

    printf("Random generated array: ");

    printArr(arr,-2);

    printf("Continue with the array? (y/n): ");

    char choice;

    choice = tolower(getchar());

    printf("\n");

    while((getchar()) != '\n');

    if (choice == 'n')

    {

        goto Generate;

    }

    printf("Do you want to visauliz     e algorithms? (y/n): ");

    choice = tolower(getchar());

    while((getchar()) != '\n');

    if (choice == 'y') visual = 1;

    printf("\nInsertion sort -->\n");

    insertion\_sort(insert\_sort, visual);

    printf("\nSelection sort -->\n");

    selection\_sort(select\_sort, visual);

    printf("\n");

    printf("Merge sort -->\n");

    mergeSort(merge\_sort, 0, size-1, visual);

    if(visual) printf("\n");

    printf("Sorted Array: ");

    printArr(merge\_sort, -1);

    printf("\nQuick sort -->\n");

    Quicksort(arr , 0, 4, visual);

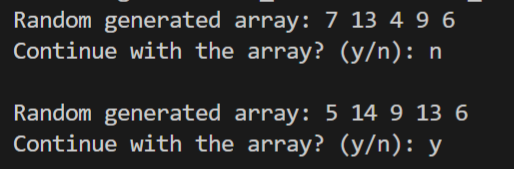
    printf("Sorted Array: ");

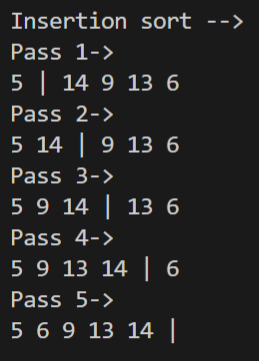
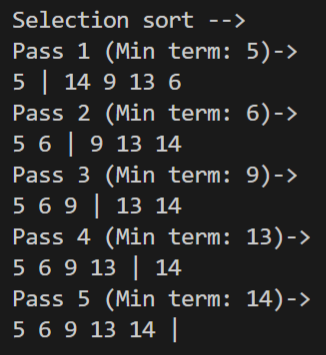
    printArr(arr, -1);

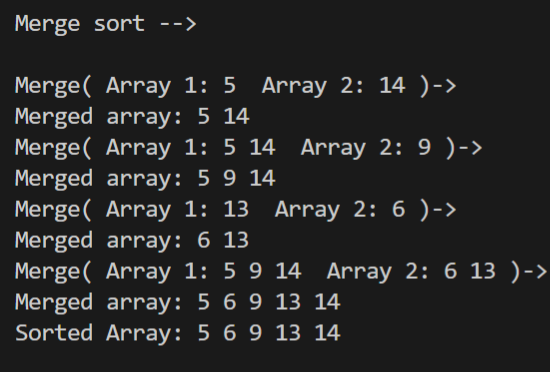
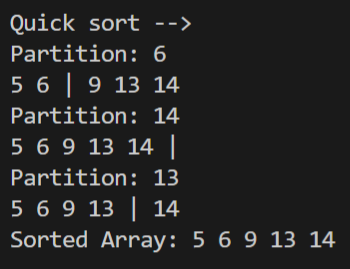
    return 0;

}

**Screenshot of compilation and execution:**

****

** **

** **