LABORATORY REPORT

Algorithm Laboratory (CS-39001)

B.Tech Program in ECSc

Submitted By

Name:- SANNIDHI DEB

Roll No: 2330044

Kalinga Institute of Industrial Technology

(Deemed to be University) Bhubaneswar, India

Autumn, 2025

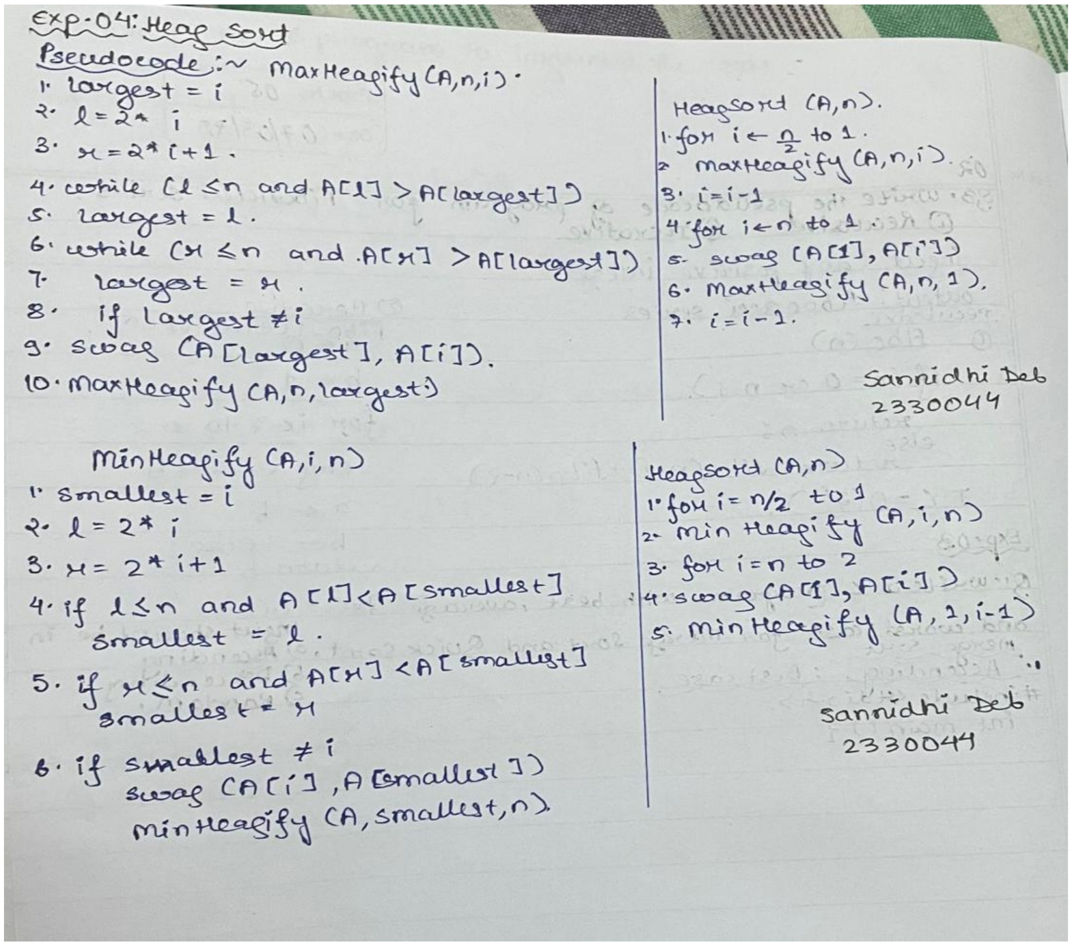
Table of Contents

Experiment Number	4.1

Experiment Title	4.1 Aim of the program:
	Define a struct person as follows:
	struct person
	{
	int id;
	char *name;
	int age;
	int height;
	int weight;
	};
	Write a menu driven program to read the data of ‘n’ students. Implement the min-heap or max-heap and its operations based on the menu options.
	Sample Input/Output:
	MAIN MENU (HEAP)
	1. Read Data
	2. Create a Min-heap based on the age
	3. Create a Max-heap based on the weight
	4. Display weight of the youngest person
	5. Insert a new person into the Min-heap
	6. Delete the oldest person
	7. Exit

Date of Experiment	21/08/2025
Date of Submission	27/08/2025

1.Algorithm:-



Exp-04: Heap Sort

Pseudocode :~ MaxHeapify (A,n,i).

```
1. largest = i
2. l = 2* i
3. r = 2* i+1.
4. while (l ≤ n and A[l] > A[largest])
5. largest = l.
6. while (r ≤ n and A[r] > A[largest])
7.    largest = r.
8.    if Largest ≠ i
9. swap (A[largest], A[i]).
10. MaxHeapify (A,n,largest)
```

```
Heapsort (A,n).
1. for i ← n/2 to 1.
2  MaxHeapify (A,n,i).
3. i=i-1
4 for i ← n to 1.
5. swap [A[1], A[i]]
6. MaxHeapify (A,n,1).
7. i=i-1.
```

Sannidhi Deb
2330044

```
    MinHeapify (A,i,n)
1. smallest = i
2. l = 2* i
3. r = 2* i+1
4. if l ≤ n and A[l] < A[smallest]
     smallest = l.
5. if r ≤ n and A[r] < A[smallest]
     smallest = r
6. if smallest ≠ i
     swap (A[i], A[smallest])
     minHeapify (A, smallest, n).
```

```
Heapsort (A,n)
1. for i = n/2 to 1
2. min Heapify (A,i,n)
3. for i = n to 2
4. swap (A[1], A[i])
5. min Heapify (A, 1, i-1)
```

Sannidhi Deb
2330044

2. Code:-

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>



struct person {

    int id;

    char name[50];

    int age;

    int height;
    int weight;
};

struct person *arr = NULL; // dynamic array of persons
int n = 0;  // number of persons

// --------- Heap Helper Functions ---------
void swap(struct person *a, struct person *b) {
    struct person temp = *a;
    *a = *b;
    *b = temp;
}

// Min-Heapify based on Age
void minHeapify(struct person arr[], int size, int i) {
    int smallest = i;
    int l = 2*i + 1;
    int r = 2*i + 2;

    if (l < size && arr[l].age < arr[smallest].age)
        smallest = l;
    if (r < size && arr[r].age < arr[smallest].age)
        smallest = r;

    if (smallest != i) {
        swap(&arr[i], &arr[smallest]);
        minHeapify(arr, size, smallest);
    }
```

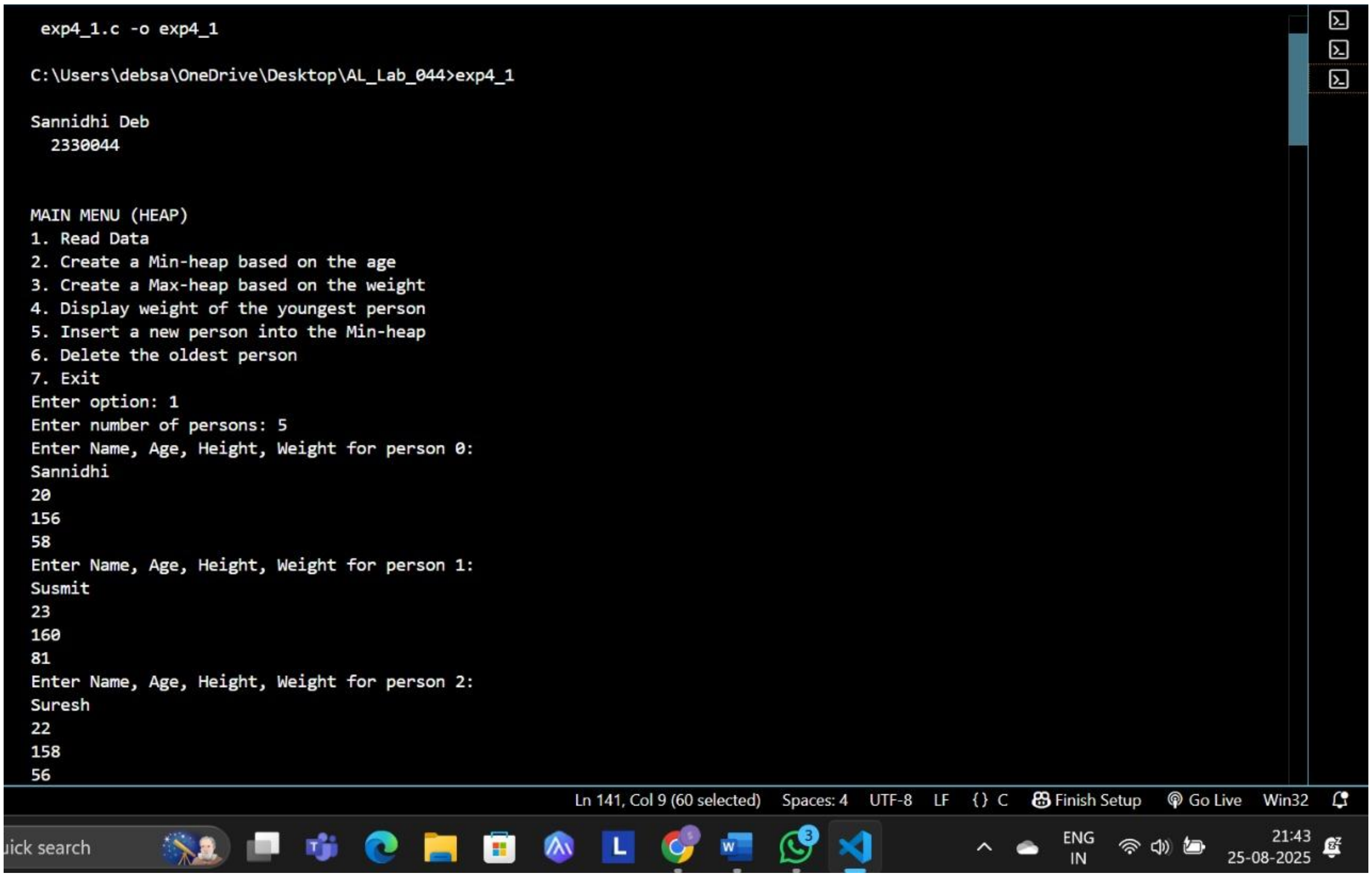
```
}

// Max-Heapify based on Weight
void maxHeapify(struct person arr[], int size, int i) {
    int largest = i;
    int l = 2*i + 1;
    int r = 2*i + 2;

    if (l < size && arr[l].weight > arr[largest].weight)
        largest = l;
    if (r < size && arr[r].weight > arr[largest].weight)
        largest = r;

    if (largest != i) {
        swap(&arr[i], &arr[largest]);
        maxHeapify(arr, size, largest);
    }
}

void buildMinHeap(struct person arr[], int size) {
    for (int i = size/2 - 1; i >= 0; i--)
        minHeapify(arr, size, i);
}

void buildMaxHeap(struct person arr[], int size) {
    for (int i = size/2 - 1; i >= 0; i--)
        maxHeapify(arr, size, i);
}
```

```
// Insert into Min-Heap (based on Age)
void insertMinHeap(struct person p) {
    n++;
    arr = realloc(arr, n * sizeof(struct person));
    arr[n-1] = p;

    int i = n - 1;
    while (i > 0 && arr[(i-1)/2].age > arr[i].age) {
        swap(&arr[i], &arr[(i-1)/2]);
        i = (i-1)/2;
    }
}

// Delete the oldest person (max age) from Min-Heap
void deleteOldest() {
    if (n == 0) {
        printf("Heap is empty!\n");
        return;
    }

    int maxAgeIdx = 0;
    for (int i = 1; i < n; i++) {
        if (arr[i].age > arr[maxAgeIdx].age)
            maxAgeIdx = i;
    }

    printf("Deleted oldest person: %s (Age %d)\n", arr[maxAgeIdx].name, arr[maxAgeIdx].age);
    arr[maxAgeIdx] = arr[n-1];

    n--;

    arr = realloc(arr, n * sizeof(struct person));

    buildMinHeap(arr, n);

}


// --------- Menu Functions ---------

void readData() {

    printf("Enter number of persons: ");

    scanf("%d", &n);


    arr = malloc(n * sizeof(struct person));


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

        arr[i].id = i;

        printf("Enter Name, Age, Height, Weight for person %d:\n", i);

        scanf("%s %d %d %d", arr[i].name, &arr[i].age, &arr[i].height, &arr[i].weight);

    }

}


void displayPersons() {

    printf("\nId\tName\tAge\tHeight\tWeight\n");

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

        printf("%d\t%s\t%d\t%d\t%d\n", arr[i].id, arr[i].name, arr[i].age, arr[i].height, arr[i].weight);

}


void displayYoungestWeight() {
    if (n == 0) {
        printf("No persons available.\n");
        return;
    }

    int minAgeIdx = 0;
    for (int i = 1; i < n; i++) {
        if (arr[i].age < arr[minAgeIdx].age)
            minAgeIdx = i;
    }

    printf("Weight of youngest person (%s, Age %d): %.2f kg\n",
           arr[minAgeIdx].name, arr[minAgeIdx].age, arr[minAgeIdx].weight * 0.453592);
}

// --------- Main ---------
int main(void) {
    int choice;
    do {
        printf("\nSannidhi Deb\n  2330044\n\n\nMAIN MENU (HEAP)\n");
        printf("1. Read Data\n");
        printf("2. Create a Min-heap based on the age\n");
        printf("3. Create a Max-heap based on the weight\n");
        printf("4. Display weight of the youngest person\n");
        printf("5. Insert a new person into the Min-heap\n");
        printf("6. Delete the oldest person\n");
        printf("7. Exit\n");
        printf("Enter option: ");
```

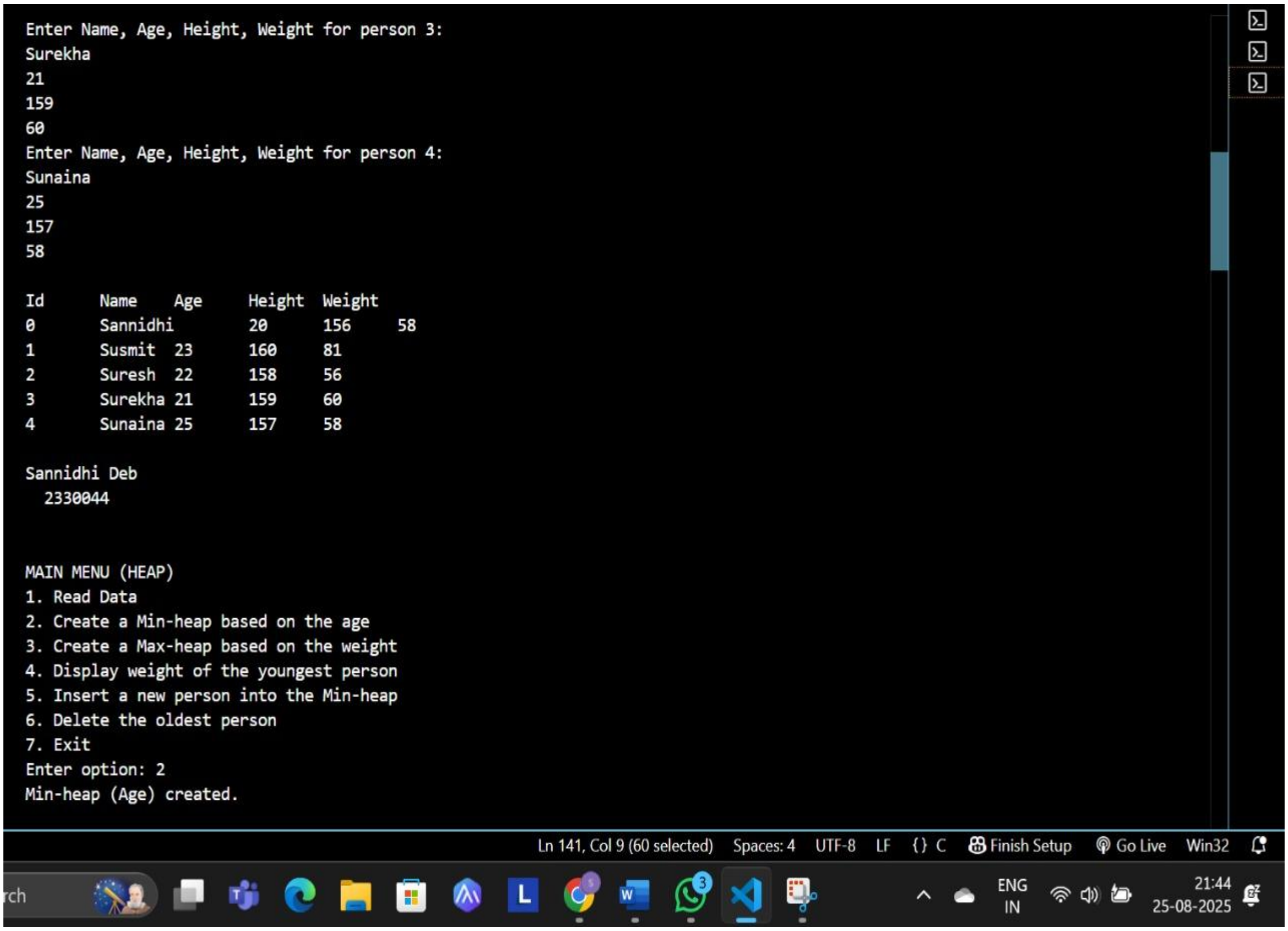
```
scanf("%d", &choice);

switch(choice) {
    case 1:
        readData();
        displayPersons();
        break;
    case 2:
        buildMinHeap(arr, n);
        printf("Min-heap (Age) created.\n");
        displayPersons();
        break;
    case 3:
        buildMaxHeap(arr, n);
        printf("Max-heap (Weight) created.\n");
        displayPersons();
        break;
    case 4:
        displayYoungestWeight();
        break;
    case 5: {
        struct person p;
        p.id = n;
        printf("Enter Name, Age, Height, Weight for new person:\n");
        scanf("%s %d %d %d", p.name, &p.age, &p.height, &p.weight);
        insertMinHeap(p);
        printf("Person inserted into Min-heap.\n");
        displayPersons();
```

```
                break;
            }
            case 6:
                deleteOldest();
                displayPersons();
                break;
            case 7:
                printf("Exiting program...\n");
                break;
            default:
                printf("Invalid option!\n");
        }
    } while(choice != 7);

    free(arr);
    return 0;
}
```

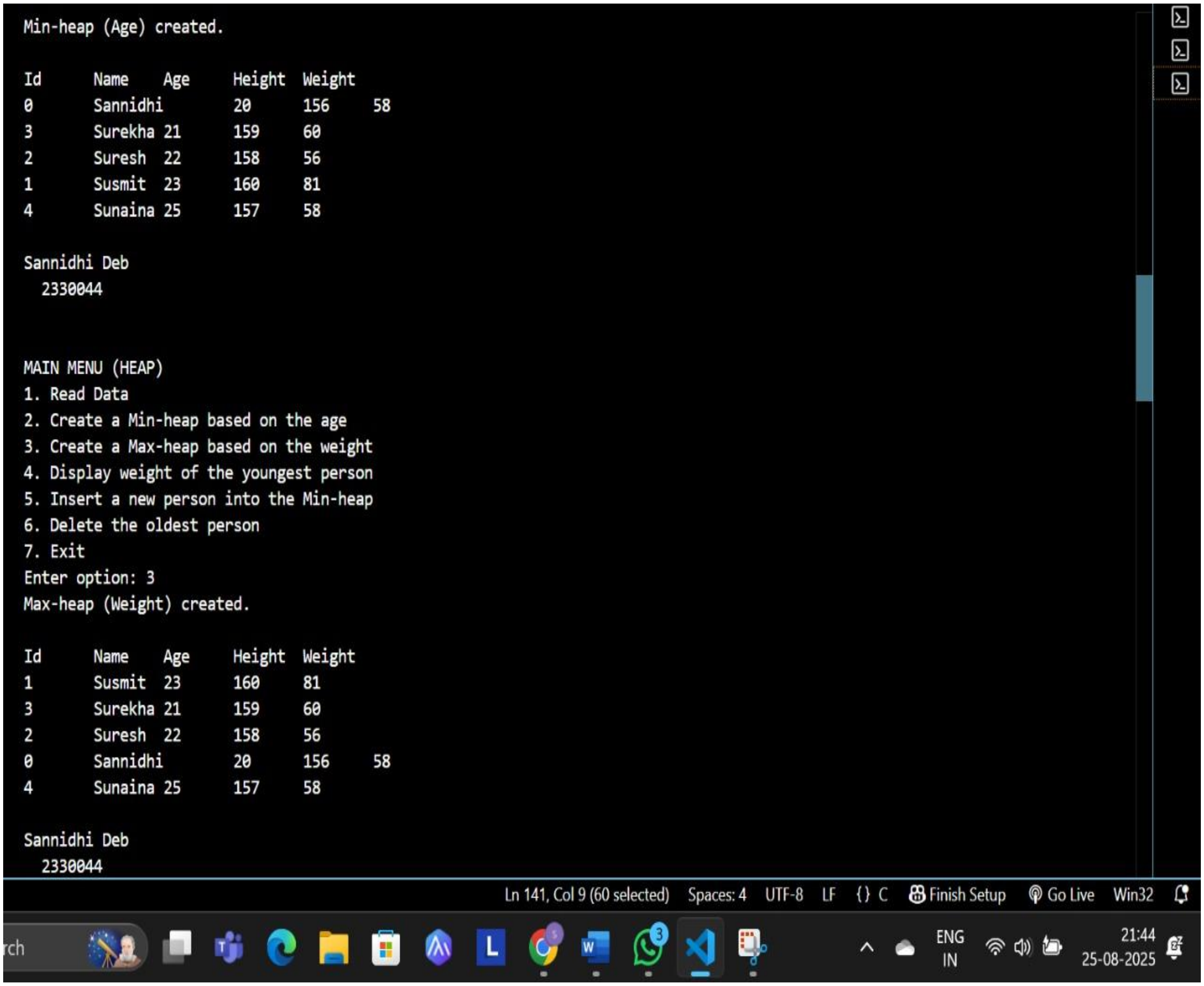
3.Results/Output:- Entire Screen Shot including Date & Time:-



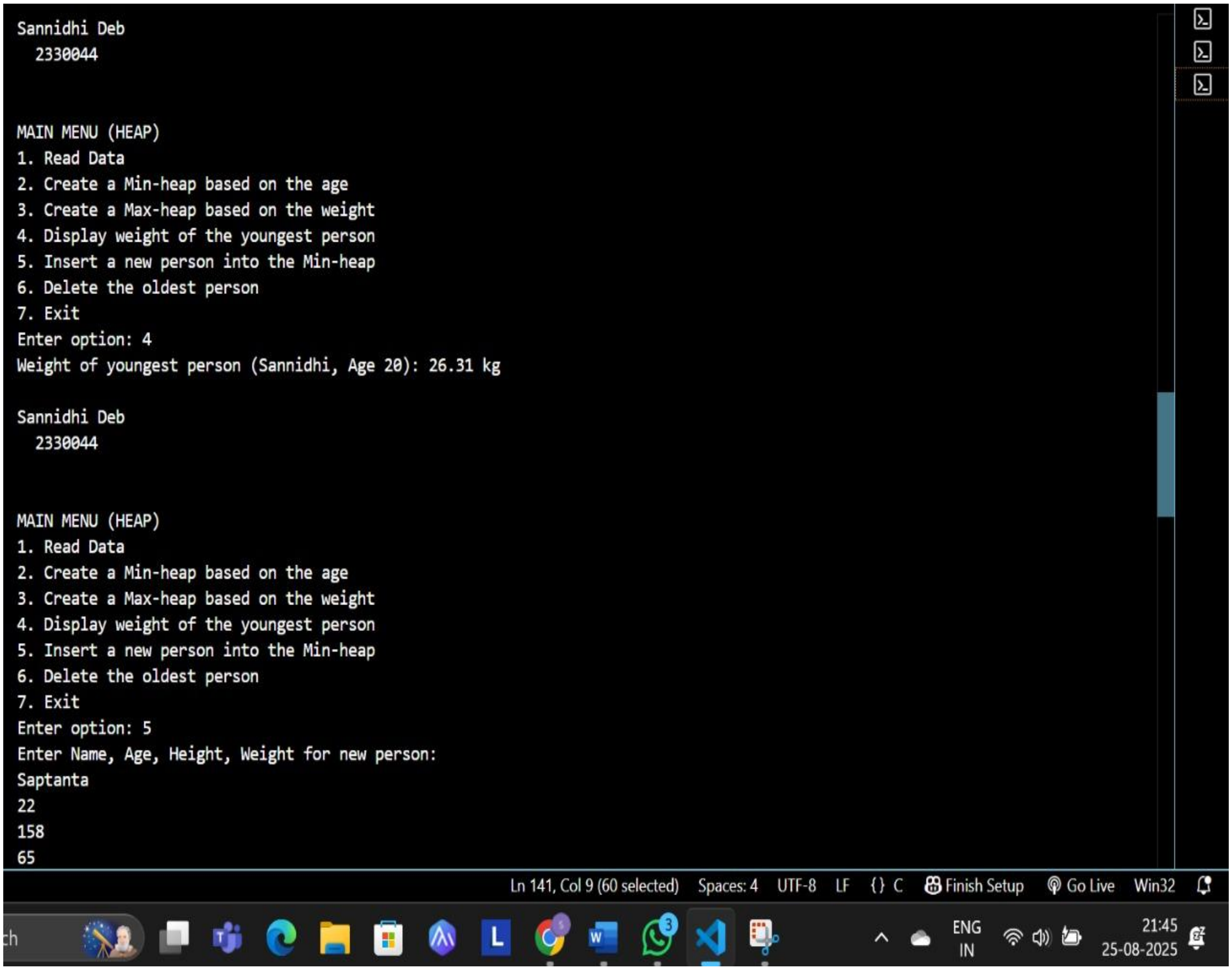
```
 exp4_1.c -o exp4_1

C:\Users\debsa\OneDrive\Desktop\AL_Lab_044>exp4_1

Sannidhi Deb
  2330044


MAIN MENU (HEAP)
1. Read Data
2. Create a Min-heap based on the age
3. Create a Max-heap based on the weight
4. Display weight of the youngest person
5. Insert a new person into the Min-heap
6. Delete the oldest person
7. Exit
Enter option: 1
Enter number of persons: 5
Enter Name, Age, Height, Weight for person 0:
Sannidhi
20
156
58
Enter Name, Age, Height, Weight for person 1:
Susmit
23
160
81
Enter Name, Age, Height, Weight for person 2:
Suresh
22
158
56
```



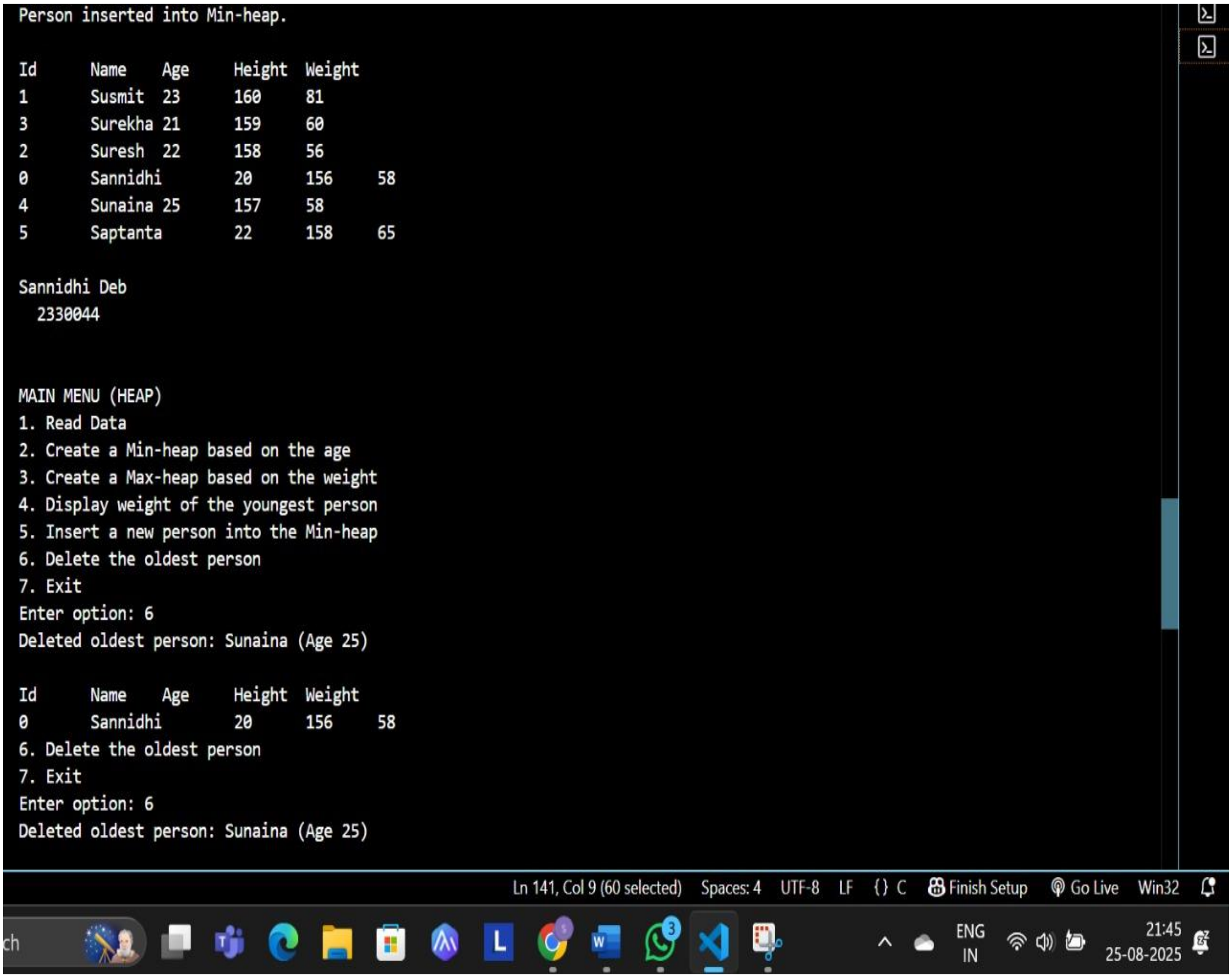
```
Enter Name, Age, Height, Weight for person 3:
Surekha
21
159
60
Enter Name, Age, Height, Weight for person 4:
Sunaina
25
157
58

Id      Name    Age     Height  Weight
0       Sannidhi        20      156     58
1       Susmit  23      160     81
2       Suresh  22      158     56
3       Surekha 21      159     60
4       Sunaina 25      157     58

Sannidhi Deb
  2330044


MAIN MENU (HEAP)
1. Read Data
2. Create a Min-heap based on the age
3. Create a Max-heap based on the weight
4. Display weight of the youngest person
5. Insert a new person into the Min-heap
6. Delete the oldest person
7. Exit
Enter option: 2
Min-heap (Age) created.
```



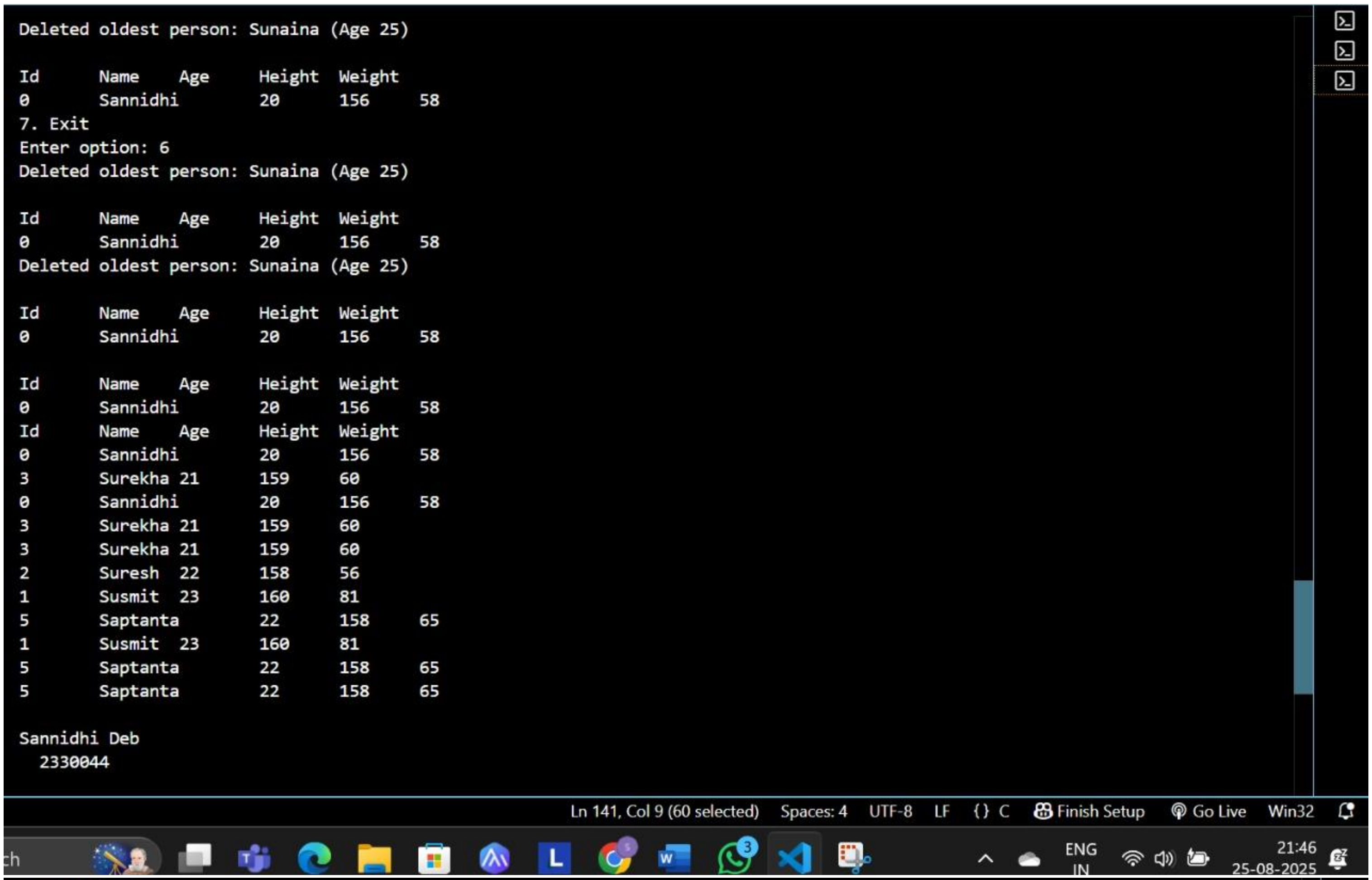
```
Min-heap (Age) created.

Id      Name    Age     Height  Weight
0       Sannidhi        20      156     58
3       Surekha 21      159     60
2       Suresh  22      158     56
1       Susmit  23      160     81
4       Sunaina 25      157     58

Sannidhi Deb
  2330044



MAIN MENU (HEAP)
1. Read Data
2. Create a Min-heap based on the age
3. Create a Max-heap based on the weight
4. Display weight of the youngest person
5. Insert a new person into the Min-heap
6. Delete the oldest person
7. Exit
Enter option: 3
Max-heap (Weight) created.

Id      Name    Age     Height  Weight
1       Susmit  23      160     81
3       Surekha 21      159     60
2       Suresh  22      158     56
0       Sannidhi        20      156     58
4       Sunaina 25      157     58

Sannidhi Deb
  2330044
```



```
Sannidhi Deb
  2330044


MAIN MENU (HEAP)
1. Read Data
2. Create a Min-heap based on the age
3. Create a Max-heap based on the weight
4. Display weight of the youngest person
5. Insert a new person into the Min-heap
6. Delete the oldest person
7. Exit
Enter option: 4
Weight of youngest person (Sannidhi, Age 20): 26.31 kg

Sannidhi Deb
  2330044


MAIN MENU (HEAP)
1. Read Data
2. Create a Min-heap based on the age
3. Create a Max-heap based on the weight
4. Display weight of the youngest person
5. Insert a new person into the Min-heap
6. Delete the oldest person
7. Exit
Enter option: 5
Enter Name, Age, Height, Weight for new person:
Saptanta
22
158
65
```



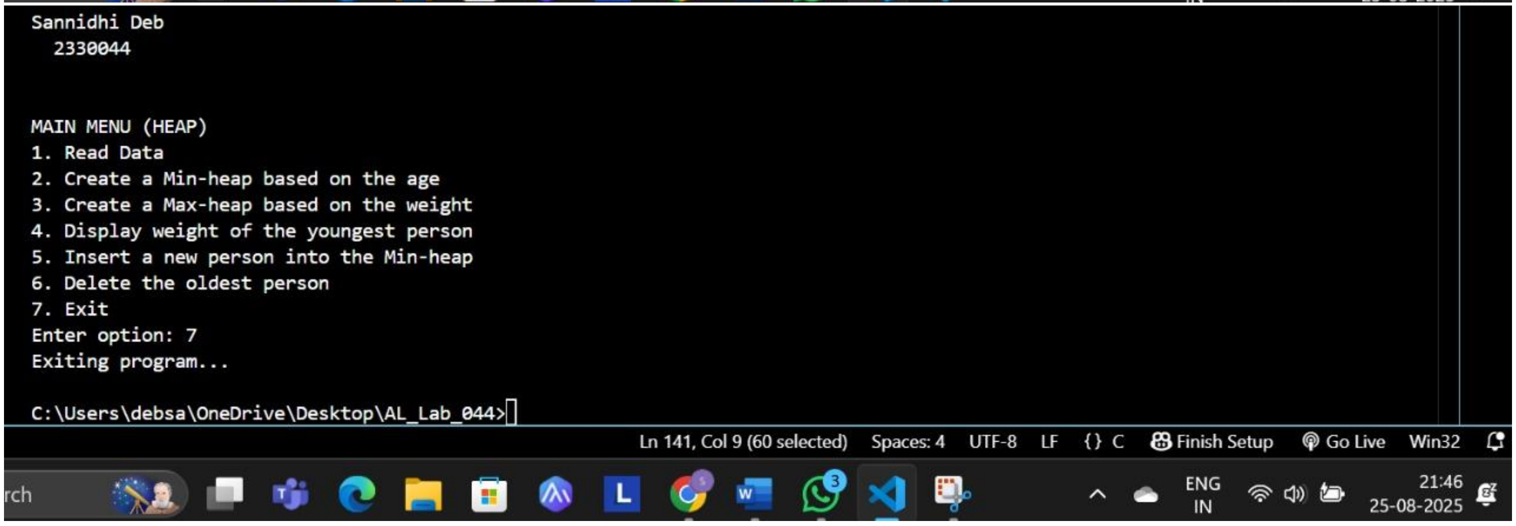
```
Person inserted into Min-heap.

Id      Name    Age     Height  Weight
1       Susmit  23      160     81
3       Surekha 21      159     60
2       Suresh  22      158     56
0       Sannidhi        20      156     58
4       Sunaina 25      157     58
5       Saptanta        22      158     65

Sannidhi Deb
  2330044


MAIN MENU (HEAP)
1. Read Data
2. Create a Min-heap based on the age
3. Create a Max-heap based on the weight
4. Display weight of the youngest person
5. Insert a new person into the Min-heap
6. Delete the oldest person
7. Exit
Enter option: 6
Deleted oldest person: Sunaina (Age 25)

Id      Name    Age     Height  Weight
0       Sannidhi        20      156     58
6. Delete the oldest person
7. Exit
Enter option: 6
Deleted oldest person: Sunaina (Age 25)
```



```
Deleted oldest person: Sunaina (Age 25)

Id      Name    Age     Height  Weight
0       Sannidhi        20      156     58
7. Exit
Enter option: 6
Deleted oldest person: Sunaina (Age 25)

Id      Name    Age     Height  Weight
0       Sannidhi        20      156     58
Deleted oldest person: Sunaina (Age 25)

Id      Name    Age     Height  Weight
0       Sannidhi        20      156     58

Id      Name    Age     Height  Weight
0       Sannidhi        20      156     58
Id      Name    Age     Height  Weight
0       Sannidhi        20      156     58
3       Surekha 21      159     60
0       Sannidhi        20      156     58
3       Surekha 21      159     60
3       Surekha 21      159     60
2       Suresh  22      158     56
1       Susmit  23      160     81
5       Saptanta        22      158     65
1       Susmit  23      160     81
5       Saptanta        22      158     65
5       Saptanta        22      158     65

Sannidhi Deb
  2330044
```

Ln 141, Col 9 (60 selected) Spaces: 4 UTF-8 LF {} C Finish Setup Go Live Win32

21:46 25-08-2025



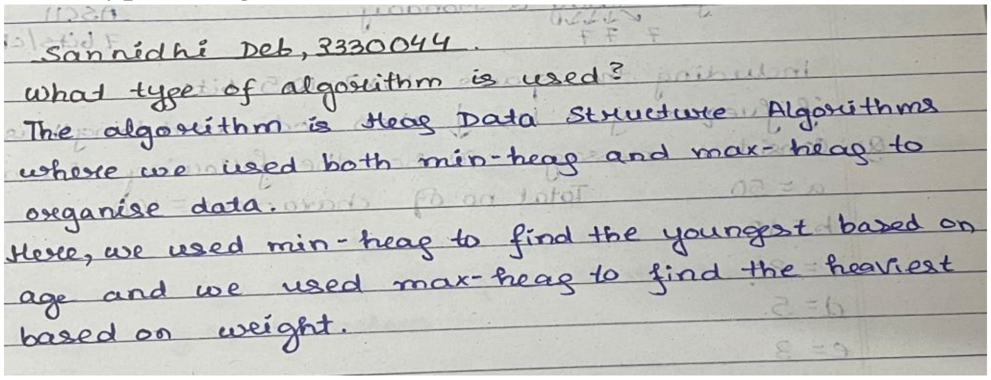
```
Sannidhi Deb
  2330044


MAIN MENU (HEAP)
1. Read Data
2. Create a Min-heap based on the age
3. Create a Max-heap based on the weight
4. Display weight of the youngest person
5. Insert a new person into the Min-heap
6. Delete the oldest person
7. Exit
Enter option: 7
Exiting program...

C:\Users\debsa\OneDrive\Desktop\AL_Lab_044>
```

Ln 141, Col 9 (60 selected) Spaces: 4 UTF-8 LF {} C Finish Setup Go Live Win32

21:46 25-08-2025

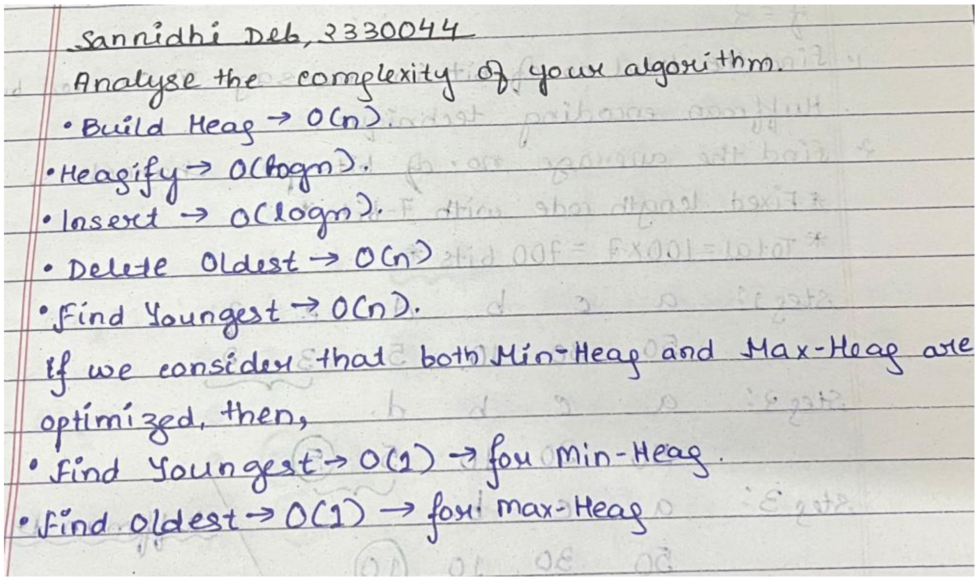
4. Remarks:-

1. What type of algorithm is used?



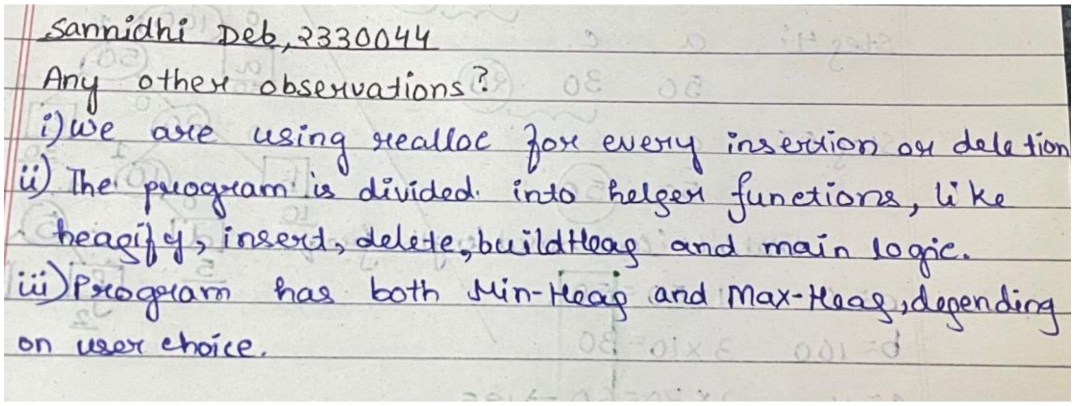
Sannidhi Deb, 2330044.
What type of algorithm is used?
The algorithm is Heap Data Structure Algorithms where we used both min-heap and max-heap to organise data.
Here, we used min-heap to find the youngest based on age and we used max-heap to find the heaviest based on weight.

2. Analyze the complexity of your algorithm.



Sannidhi Deb, 2330044
Analyse the complexity of your algorithm.
• Build Heap → O(n).
• Heapify → O(logn).
• Insert → O(logn).
• Delete Oldest → O(n)
• Find Youngest → O(n).
If we consider that both Min-Heap and Max-Heap are optimized, then,
• Find Youngest → O(1) → for Min-Heap.
• Find Oldest → O(1) → for Max-Heap

3.Any other observations?



Sannidhi Deb, 2330044

Any other observations?

i) We are using realloc for every insertion or deletion

ii) The program is divided into helper functions, like heapify, insert, delete, buildHeap and main logic.

iii) Program has both Min-Heap and Max-Heap, depending on user choice.

5. Conclusion:-

The program helped in understanding how heaps can be used for priority-based operations. Using min-heap and max-heap on student data made it easier to insert, delete, and quickly access required information like youngest or heaviest person.

Sannidhi Deb
(2330044)

Signature of the FIC

Sannidhi Deb

(Name of the FIC)