DATA STRUCTURE MINI PROJECT

Project Title:

Student Birthday Sorter using Heap Sort

Presented
by
Sathyan.B

1. Introduction:

To perform this task efficiently, we can use a Heap Sort algorithm.
Heap Sort is a comparison-based sorting technique based on a binary heap data structure. It divides the data into two parts — a heap and the sorted list — and repeatedly removes the largest (or smallest) element from the heap until all elements are sorted.

In this project, the Heap Sort algorithm is used to arrange student birthdays in ascending order (oldest to youngest) or descending order (youngest to oldest).

2. Objective:

To design a program that sorts students' birthdays efficiently.

To demonstrate the use of the Heap Sort algorithm in real-world data organization.

To improve understanding of sorting algorithms and their performance.

3. Algorithm Used HeapSort:

Heap Sort Steps:

1. Build a max heap from the given data.

2. Swap the first element (largest) with the last element.

3. Reduce the heap size by one and heapify the root.

4. Repeat the process until all elements are sorted.

Heap Sort can also be adapted as a min-heap for ascending sorting (smallest to largest).

4. C Program:

```
#include <stdio.h>
#include <string.h>

struct Student {
    char name[50];
int day, month, year;
};

void swap(struct Student *a, struct Student *b) {
struct Student temp = *a;
*a = *b;
*b = temp;
}

int compare(struct Student a, struct Student b) {
if (a.year != b.year)
    return a.year > b.year;
if (a.month != b.month)
    return a.month > b.month;
return a.day > b.day;
}
```

```
void heapify(struct Student arr[], int n, int i) {
int largest = i;
int left = 2 * i + 1;
int right = 2 * i + 2;

if (left < n && compare(arr[left], arr[largest]))
largest = left;

if (right < n && compare(arr[right], arr[largest]))
largest = right;

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

void heapSort(struct Student arr[], int n) {
for (int i = n / 2 - 1; i >= 0; i--)
heapify(arr, n, i);

for (int i = n - 1; i > 0; i--) {
swap(&arr[0], &arr[i]);
heapify(arr, i, 0);
}
}
```

```
int main() {
int n;
printf("Enter number of students: ");
scanf("%d", &n);

struct Student s[n];

for (int i = 0; i < n; i++) {
printf("\nEnter name: ");
scanf("%s", s[i].name);
printf("Enter birthday (DD MM YYYY): ");
scanf("%d %d %d", &s[i].day, &s[i].month, &s[i].year);
}

heapSort(s, n);

printf("\nSorted List of Students by Birthday:\n");
for (int i = 0; i < n; i++) {
printf("%s - %02d/%02d/%04d\n",
s[i].name, s[i].day, s[i].month, s[i].year);
}

return 0;
}
```

5. Example Output

```
        Enter number of students: 3
        Enter name: Arjun
        Enter birthday : 25 05 2002
        Enter name: Priya
        Enter birthday :14 11 20
        Enter name: Karan
        Enter birthday: 03 03 2003
    Sorted List of Student Birthday:
        Priya - 14/11/2001
        Arjun - 25/05/2002
        Karan - 03/03/2003
```

6.Advantages:

1. Makes the source code clean and readable.

2. Useful for preprocessing before compilation.

3 . Saves manual effort by automatically removing all comments.

Disadvantages:

1. May remove comments that contain important notes or explanations.

2. Does not handle nested or complex comment patterns well.

3. Works only for specific languages (like C/C++), not all file types.

7.Real-World Example:

In college management systems,
Heap Sort can be used to:

Sort students by birthday,
marks, or registration number.

Prepare monthly birthday lists.

Rank students in exams or contests.

Manage event schedules where dates must be sorted.

8.Conclusion:

1. Efficiently sorts student birthdays using Heap Sort
.
2. Improves understanding of heap data structure.
3. Useful for real-world student data management.
4. Can be enhanced with more sorting features.