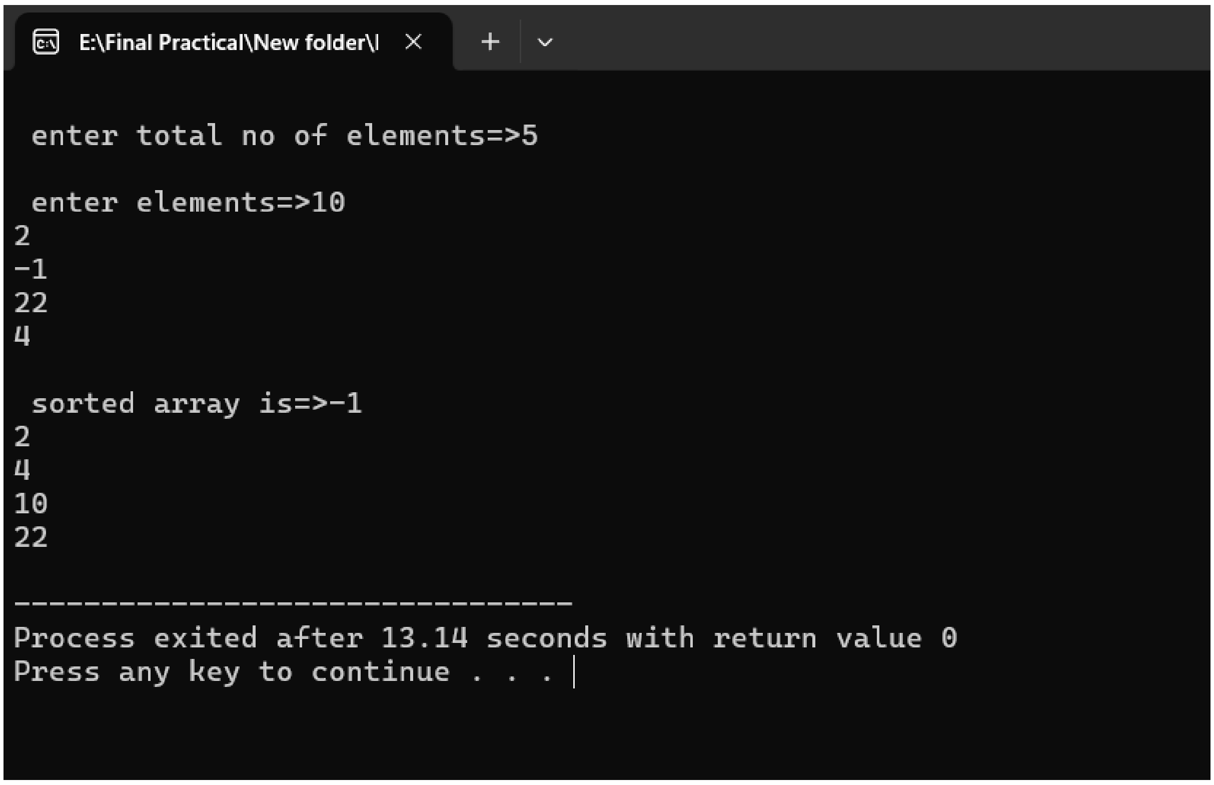
HPC Practical 4

```
#include<iostream>
#include<stdlib.h>
#include<omp.h>
using namespace std;
void mergesort(int a[],int i,int j);
void merge(int a[],int i1,int j1,int i2,int j2);
void mergesort(int a[],int i,int j)
{
    int mid;
    if(i<j)
    {
        mid=(i+j)/2;
        {
            {
                mergesort(a,i,mid);
            }
            {
                mergesort(a,mid+1,j);
            }
        }
        merge(a,i,mid,mid+1,j);
    }
}
void merge(int a[],int i1,int j1,int i2,int j2)
{
    int temp[1000];
    int i,j,k;
    i=i1;
    j=i2;
    k=0;
    while(i<=j1 && j<=j2)
    {
        if(a[i]<a[j])
        {
            temp[k++]=a[i++];
        }
        else
        {
            temp[k++]=a[j++];
    }
    }
    while(i<=j1)
    {
        temp[k++]=a[i++];
    }
    while(j<=j2)
    {
        temp[k++]=a[j++];
    }
    for(i=i1,j=0;i<=j2;i++,j++)
    {
        a[i]=temp[j];
    }
}
int main()
{
    int *a,n,i;
    cout<<"\n enter total no of elements=>";
    cin>>n;
    a= new int[n];
    cout<<"\n enter elements=>";
    for(i=0;i<n;i++)
    {
        cin>>a[i];
    }
    mergesort(a, 0, n-1);
    cout<<"\n sorted array is=>";
    for(i=0;i<n;i++)
    {
        cout<<"\n"<<a[i];
    }
    return 0;
}
```



E:\Final Practical\New folder\

```
 enter total no of elements=>5

 enter elements=>10
2
-1
22
4

 sorted array is=>-1
2
4
10
22

--------------------------------
Process exited after 13.14 seconds with return value 0
Press any key to continue . . .
```

Let's break down the Merge Sort step by step for the given example:

1. **Input**:
    - Total number of elements: 5
    - Elements: 10, 2, -1, 22, 4
2. **Initial Array**: [10, 2, -1, 22, 4]
3. **Splitting Phase**:
    - The array is divided into two halves:
        - Left half: [10, 2, -1]
        - Right half: [22, 4]
4. **Recursive Sorting**:

- Left half: [10, 2, -1]
    - Split into [10] and [2, -1]
        - Merge [10] and [2, -1] to get [2, -1, 10]
- Right half: [22, 4]
    - Split into [22] and [4]
        - Merge [22] and [4] to get [4, 22]

5. **Merging Phase**:
    - Merge the two sorted halves [2, -1, 10] and [4, 22] to get the final sorted array:
        - Compare the first elements of each half (2 and 4). Since 2 < 4, add 2 to the result array.
        - Compare the next elements (2 and 4 again). Since 4 > -1, add -1 to the result array.
        - Compare the next elements (2 and 4 again). Since 4 > 10, add 10 to the result array.
        - Now, only 4 and 22 are left. Add them to the result array: [2, -1, 10, 4, 22]
6. **Final Sorted Array**: [-1, 2, 4, 10, 22]

So, the sorted array using Merge Sort for the given input is [-1, 2, 4, 10, 22].

Theory for Practical

📌 What is Merge sort

Merge Sort is a sorting algorithm that follows the divide-and-conquer strategy to sort a list of elements. It works as follows:

1. **Divide**: The unsorted list is divided into two sublists of roughly equal size.
2. **Conquer**: Each sublist is recursively sorted.
3. **Combine**: The sorted sublists are merged to produce a single sorted list.

Merge Sort is a stable sort, which means that it preserves the relative order of equal elements in the sorted output. It has a time complexity of O(n log n), making it efficient for sorting large lists.

📌 What is Parallel Merge Sort

Parallel Merge Sort is a parallelized version of the Merge Sort algorithm that takes advantage of multiple processors or cores to sort a list of elements more quickly. It follows the same basic steps as the sequential Merge Sort but uses parallel processing to divide the work among different processors or threads.

Here's how Parallel Merge Sort works:

1. **Divide**: The list of elements is divided into smaller sublists.
2. **Conquer (Parallel Sorting)**: Each sublist is sorted independently using a parallel sorting algorithm. This can be done concurrently by different processors or threads.
3. **Merge (Parallel Merging)**: Once all sublists are sorted, they are merged back together to form the final sorted list. This merging process can also be parallelized to some extent, depending on the implementation.
4. **Repeat if Necessary**: If the list is not fully sorted after the first pass, the process is repeated until the entire list is sorted.

Parallel Merge Sort can be more efficient than sequential Merge Sort for large lists, especially on systems with multiple processors or cores. However, it requires careful management of parallelism to ensure that the sorting and merging processes are synchronized correctly.

📌 What is OpenMP

OpenMP is a programming interface that helps you write programs that can run faster on computers with multiple processors or cores. It does this by allowing you to divide your program into parts that can be done at the same time by different processors. This can make your program run faster and more efficiently.

📌 Explain Parallel BFS to a 5-year-old kid

Imagine you have a big pile of toys that you want to organize neatly. Parallel Merge Sort is like having many friends help you with this task.

1. **Divide**: You and your friends first divide the pile into smaller piles so each of you can work on a smaller set of toys.
2. **Sort (Parallel Sorting)**: Each of you then sorts your own pile of toys separately, putting them in order from smallest to biggest.
3. **Merge (Parallel Merging)**: Once everyone has sorted their toys, you start merging the sorted piles back together. You do this by comparing the toys in each pile and putting them back together in the correct order.
4. **Repeat if Necessary**: If the pile is still not completely sorted, you can repeat the process until everything is sorted.

By working together, you and your friends can sort the toys much faster than if you were doing it alone. This is how Parallel Merge Sort helps us sort things quickly using teamwork!

Viva Questions ❓ ❓

1. **What is Parallel Merge Sort?**
    - Answer: Parallel Merge Sort is a parallelized version of the Merge Sort algorithm that uses multiple processors or cores to sort a list of elements more efficiently.
2. **How does Parallel Merge Sort work?**

- Answer: Parallel Merge Sort divides the list into smaller sublists, which are then independently sorted using parallel processing. The sorted sublists are then merged back together to form the final sorted list.

3. **What is the advantage of Parallel Merge Sort over regular Merge Sort?**
    - Answer: Parallel Merge Sort can be faster than regular Merge Sort for large lists because it can take advantage of multiple processors or cores to sort the list more quickly.
4. **What are the challenges of implementing Parallel Merge Sort?**
    - Answer: One challenge is ensuring that the sorting and merging processes are synchronized correctly to avoid errors. Another challenge is efficiently dividing the work among multiple processors or cores.
5. **Can Parallel Merge Sort be used to sort any type of data?**
    - Answer: Yes, Parallel Merge Sort can be used to sort any type of data for which a comparison operation is defined, such as numbers or strings.
6. **Is Parallel Merge Sort a stable sorting algorithm?**
    - Answer: Yes, Parallel Merge Sort is a stable sorting algorithm, meaning that it preserves the relative order of equal elements in the sorted output.
7. **What is the time complexity of Parallel Merge Sort?**
    - Answer: The time complexity of Parallel Merge Sort is O(n log n), where n is the number of elements in the list. This makes it efficient for sorting large lists.
8. **How does Parallel Merge Sort compare to other parallel sorting algorithms?**
    - Answer: Parallel Merge Sort is known for its simplicity and ease of implementation compared to other parallel sorting algorithms. It is also efficient and can be adapted to different parallel computing architectures.
9. **Can Parallel Merge Sort be used on a single-core processor?**
    - Answer: Yes, Parallel Merge Sort can still be used on a single-core processor, but it may not provide significant speedup compared to

regular Merge Sort due to the lack of parallel processing capabilities.

10. **In what scenarios would you choose to use Parallel Merge Sort?**
    - Answer: Parallel Merge Sort is suitable for sorting large lists of data where the use of multiple processors or cores can significantly speed up the sorting process.