Lab -

DIP : Design and Analysis of Algorithm

AIM : Implement the following Divide & Conquer Problems

(1) Merge Sort

(2) Quick Sort

Name : Ambalia Harshit

Department : Computer Engineering (M.Tech sem II)

Roll no : MT001

Date : Jan 2024

Program 01 : Merge Sort(Recursion)

- Description :

  The objective of merge sort using the divide and conquer approach is to sort an array of elements in ascending or descending order.

  The divide and conquer approach is used to break down the array into smaller sub-arrays, sort each sub-array individually, and then merge the sub-arrays back together in the correct order.

  The main advantage of merge sort is that it is a stable sorting algorithm, meaning that it preserves the relative order of elements with equal values.

- Algorithm :

ALGORITHM-MERGE SORT

1. If p<r
2. Then q → (p+ r)/2
3. MERGE-SORT (A, p, q)
4. MERGE-SORT (A, q+1,r)
5. MERGE (A, p, q, r)

FUNCTIONS: MERGE (A, p, q, r)

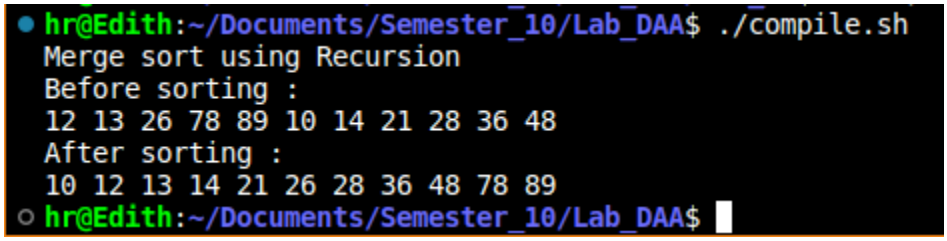
1. n 1 = q-p+1
2. n 2= r-q
3. create arrays [1.....n 1 + 1] and R [1.....n 2 +1]
4. for i ← 1 to n 1
5. do [i] ← A [p+ i-1]
6. for j ← 1 to n2
7. do R[j] ← A[q + j]
8. L [n 1+ 1] ← ∞
9. R[n 2+ 1] ← ∞
10. I ← 1
11. J ← 1
12. For k ← p to r
13. Do if L [i] ≤ R[j]
14. then A[k] ← L[i]
15. i ← i +1
16. else A[k] ← R[j]
17. j ← j+

- Code :

```
// Simple Merge sort - Recursion

#include <stdio.h>
#include <stdlib.h>
#include "/home/hr/Documents/Semester_10/Lab_DAA/HRA.h"

// void merge(int list[], int low, int mid, int high);

// void my_merge_sort_recursion(int arr[], int p, int r) {
//     int q;
//     if (p < r) {
//         q = (p + r) / 2;
//         merge_sort_recursion(arr, p, q);
//         merge_sort_recursion(arr, q + 1, r);
//         merge_for_recursion(arr, p, q, r);
//     }
// }

// void merge_for_recursion(int list[], int low, int mid, int high) {
//     int n1 = mid-low+1;
//     int n2 = high-mid;
//     int arr1[n1], arr2[n2];
//     for (int i=0;i<n1;i++) {
//         arr1[i] = list[low+i];
//     }
//     for (int i=0;i<n2;i++) {
//         arr2[i] = list[mid+i+1];
//     }

//     int i=0,j=0,k=low;
//     while (i < n1 && j < n2) {
//         if (arr1[i] <= arr2[j]) {
//             list[k] = arr1[i];
//             i++;
//         } else {
//             list[k] = arr2[j];
//             j++;
//         }
```

```
//         k++;
//     }
//     while (i < n1) {
//         list[k] = arr1[i];
//         i++;
//         k++;
//     }
//     while (j < n2) {
//         list[k] = arr2[j];
//         j++;
//         k++;
//     }
// }

int main() {
   int list[] = {12, 13, 26, 78, 89, 10, 14, 21, 28, 36, 48};
   int length = 11;

   printf("Merge sort using Recursion\n");

   printf("Before sorting :\n");
   for (int i=0;i<length;i++)
       printf("%d ", list[i]);
   printf("\n");

   //
   my_merge_sort_recursion(list, 0, length-1);
   //

   printf("After sorting :\n");
   for (int i=0;i<length;i++)
       printf("%d ", list[i]);
   printf("\n");

   return 0;
}
```

- Output Screen-shots :



Program 02 : Quick Sort(Recursion)

- Description :

  The objective of quick sort using the divide and conquer approach is to sort an array of elements in ascending or descending order.

  The divide and conquer approach is used to partition the array into smaller sub-arrays, and to sort each partition individually.

  The main advantage of quick sort is its efficiency for large data sets, and its ability to sort the data in place, meaning that it doesn't use extra memory.

  Additionally, it can also be easily adapted to sort data in different ways, such as sorting it in descending order. It's worth noting that quick sort is not a stable sorting algorithm, which means that it can change the relative order of elements with equal values.

- Algorithm :

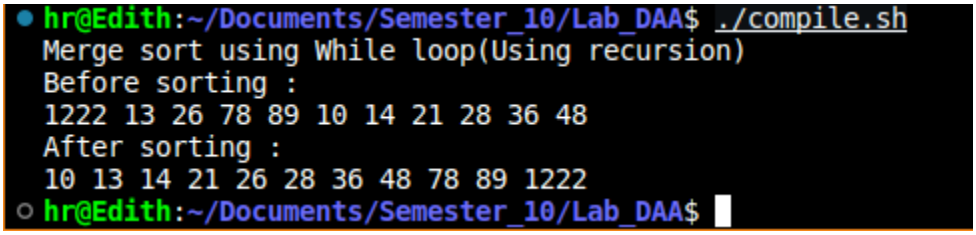
```
QUICKSORT (array A, int m, int n)
    1 if (n > m)
    2 then
    3 i ← a random index from [m,n]
    4 swap A [i] with A[m]
    5 o ← PARTITION (A, m, n)
    6 QUICKSORT (A, m, o - 1)
    7 QUICKSORT (A, o + 1, n)

PARTITION (array A, int m, int n)
    1 x ← A[m]
    2 o ← m
    3 for p ← m + 1 to n
    4 do if (A[p] < x)
    5 then o ← o + 1
    6 swap A[o] with A[p]
    7 swap A[m] with A[o]
    8 return o
```

- Code :

```
// Quick sort - Using reccursion(Last element as PIVOT)

#include <stdio.h>
#include "/home/hr/Documents/Semester_10/Lab_DAA/HRA.h"

// void swap_two_variable(int *a, int *b) {
//   int t = *a;
//   *a = *b;
//   *b = t;
// }

// int temp_partition(int arr[], int low, int high) {
//     int pivot = arr[high];
//     int o = low-1;
//     for (int i=low;i<high;i++) {
//         if(arr[i]<=pivot) {
//             o = o+1;
//             swap_two_variable(&arr[o], &arr[i]);
//         }
//     }
//     swap_two_variable(&arr[o+1], &arr[high]);
//     return (o+1);
// }

// void my_quick_sort_recursion(int arr[], int low, int high) {
//     if (low<high) {
//         int pivot = temp_partition(arr, low, high);
//         my_quick_sort_recursion(arr, low, pivot-1);
//         my_quick_sort_recursion(arr, pivot+1, high);
//     }
// }

int main() {
   int list[] = {1222, 13, 26, 78, 89, 10, 14, 21, 28, 36, 48};
   int length = 11;

   printf("Merge sort using While loop(Using recursion)\n");
```

```
    printf("Before sorting :\n");
    for (int i=0;i<length;i++)
        printf("%d ", list[i]);
    printf("\n");

    //
    my_quick_sort_recursion(list, 0, length-1);
    //

    printf("After sorting :\n");
    for (int i=0;i<length;i++)
        printf("%d ", list[i]);
    printf("\n");

    return 0;
}
```

- Output Screen-shots :



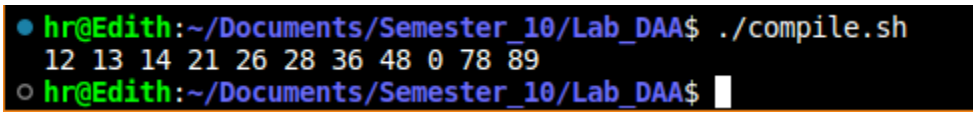
Extra 01 : Merge function(Using while loop)

- Code :

```c
// Simple merge - While loop

#include <stdio.h>
#include <stdlib.h>

void merge(int list[], int low, int mid, int high) {
   int n1 = mid-low+1;
   int n2 = high-mid;
   int arr1[n1], arr2[n2];
   for (int i=0;i<n1;i++) {
       arr1[i] = list[low+i];
   }
   for (int i=0;i<n2;i++) {
       arr2[i] = list[mid+i+1];
   }

   int i=0,j=0,k=low;
   while (i < n1 && j < n2) {
       if (arr1[i] <= arr2[j]) {
           list[k] = arr1[i];
           i++;
       } else {
           list[k] = arr2[j];
           j++;
       }
       k++;
   }

   while (i < n1) {
       list[k] = arr1[i];
       i++;
       k++;
   }

   while (j < n2) {
```

```
        list[k] = arr2[j];
        j++;
        k++;
    }

    for(int i=0;i<high;i++) {
        printf("%d ", list[i]);
    }
    printf("\n");
}

int main() {
    int list[] = {12, 13, 26, 78, 89, 10, 14, 21, 28, 36, 48};
    int length = 11;
    merge(list, 1, length/2, length);
}
```

- Output Screen-Shots :



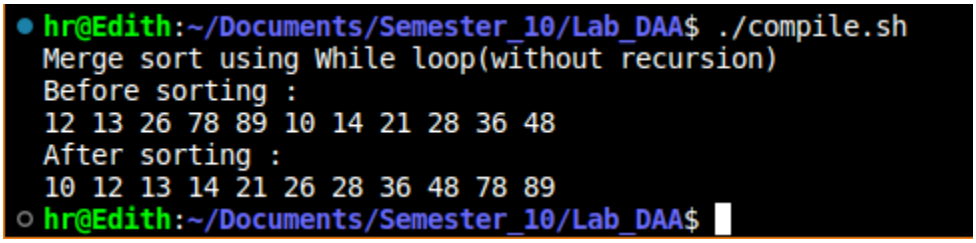
Extra 02 : Merge sort(Using while loop)

- Code :

```
// Simple Merge sort - While loop

#include <stdio.h>
#include "/home/hr/Documents/Semester_10/Lab_DAA/HRA.h"

// void merge(int list[], int low, int mid, int high);

// void my_merge_sort_while(int arr[], int length) {
//     for (int i=1;i<=length-1;i=2*i) {
//         for (int left_start=0;left_start<length-1;left_start+=2*i) {
//             int mid = left_start+i-1;
//             int right_end = (left_start+2*i-1<length-1) ?
(left_start+2*i-1) : (length-1);
//             merge(arr, left_start, mid,right_end);
//         }
//     }
// }

// void merge(int list[], int low, int mid, int high) {
//     int n1 = mid-low+1;
//     int n2 = high-mid;
//     int arr1[n1], arr2[n2];
//     for (int i=0;i<n1;i++) {
//         arr1[i] = list[low+i];
//     }
//     for (int i=0;i<n2;i++) {
//         arr2[i] = list[mid+i+1];
//     }

//     int i=0,j=0,k=low;
//     while (i < n1 && j < n2) {
//         if (arr1[i] <= arr2[j]) {
//             list[k] = arr1[i];
//             i++;
//         } else {
```

```
//             list[k] = arr2[j];
//             j++;
//         }
//         k++;
//     }

//     while (i < n1) {
//         list[k] = arr1[i];
//         i++;
//         k++;
//     }

//     while (j < n2) {
//         list[k] = arr2[j];
//         j++;
//         k++;
//     }
// }

int main() {
   int list[] = {12, 13, 26, 78, 89, 10, 14, 21, 28, 36, 48};
   int length = 11;

   printf("Merge sort using While loop(without recursion)\n");
   printf("Before sorting :\n");
   for (int i=0;i<length;i++)
       printf("%d ", list[i]);
   printf("\n");

   //
   my_merge_sort_while(list, length);
   //

   printf("After sorting :\n");
   for (int i=0;i<length;i++)
       printf("%d ", list[i]);
   printf("\n");

   return 0;
}
```

- Output Screen-Shot :



Extra 03 : Quick sort(Using while loop)

```
// Quick sort - Using while loop(Last element as PIVOT)

#include <stdio.h>
#include "/home/hr/Documents/Semester_10/Lab_DAA/HRA.h"

// void swap_two_variable(int *a, int *b) {
//     int t = *a;
//     *a = *b;
//     *b = t;
// }

// int temp_partition(int arr[], int low, int high) {
//     int pivot = arr[high];
//     int o = low-1;
//     for (int i=low;i<high;i++) {
//         if(arr[i]<=pivot) {
//             o = o+1;
//             swap_two_variable(&arr[o], &arr[i]);
//         }
//     }
//     swap_two_variable(&arr[o+1], &arr[high]);
//     return (o+1);
// }

// void my_quick_sort(int arr[], int low, int high) {
//     int stack[high-low+1];
//     int top = -1;
//     stack[++top] = low;
//     stack[++top] = high;
//     while (top >= 0) {
//         high = stack[top--];
//         low = stack[top--];
//         int pivot = temp_partition(arr, low, high);
//         if (pivot-1>=low) {
//             stack[++top] = low;
//             stack[++top] = pivot-1;
//         }
```

```
//         if (pivot+1<high) {
//             stack[++top] = pivot+1;
//             stack[++top] = high;
//         }
//     }
// }

int main() {
   int list[] = {12, 13, 26, 78, 89, 10, 14, 21, 28, 36, 48};
   int length = 11;

   printf("Quick sort using While loop\n");

   printf("Before sorting :\n");
   for (int i=0;i<length;i++)
       printf("%d ", list[i]);
   printf("\n");

   //
   // my_quick_sort(list, 0, length-1);
   void my_quick_sort_while(int arr[], int low, int high);
   //

   printf("After sorting :\n");
   for (int i=0;i<length;i++)
       printf("%d ", list[i]);
   printf("\n");

   return 0;
}
```

- Output Screen-Shot :