Merge Sort and Analysis
--- Analyzing Recursive Programs

Debdeep Mukhopadhyay
IIT Madras
Why are we dealing with merge sort in this course?

• It is a powerful application of Divide and Conquer technique in problem solving.
• Also radically different from insertion, selection and bubble sort.
• We can now apply the techniques and compare the performance of this sort with the previous ones.
Merging

• Produce a sorted list from two \textit{sorted} lists
• A simple way is to examine from the front.
• At each step find the smaller of the two elements at the current fronts
• Choose that element as the next element of the merged list.
• Remove the chosen element from its list, exposing the next element as the now first element.
### Example of Merging Iteratively

<table>
<thead>
<tr>
<th>L1</th>
<th>L2</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,2,7,7,9</td>
<td>2,4,7,8</td>
<td>Empty</td>
</tr>
<tr>
<td>2,7,7,9</td>
<td>2,4,7,8</td>
<td>1</td>
</tr>
<tr>
<td>7,7,9</td>
<td>2,4,7,8</td>
<td>1,2</td>
</tr>
<tr>
<td>7,7,9</td>
<td>4,7,8</td>
<td>1,2,2</td>
</tr>
<tr>
<td>7,7,9</td>
<td>7,8</td>
<td>1,2,2,4</td>
</tr>
<tr>
<td>7,9</td>
<td>7,8</td>
<td>1,2,2,4,7</td>
</tr>
<tr>
<td>9</td>
<td>7,8</td>
<td>1,2,2,4,7,7</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>1,2,2,4,7,7,7</td>
</tr>
<tr>
<td>Empty</td>
<td>Empty</td>
<td>1,2,2,4,7,7,7,8</td>
</tr>
<tr>
<td></td>
<td>Empty</td>
<td>1,2,2,4,7,7,7,8,9</td>
</tr>
</tbody>
</table>
Implementation of Merge Sort (MS)

- It is easier to conceive in terms of a Linked List (LL)
- We have seen LLs in the lab.
- It is represented by a node (cell), which has two components: one info and the other pointer to the next element.

```c
struct node{
    int info;
    struct node *next;
};
typedef struct node *NODEPTR;
```

Definition of the nodes in C
Inserting elements into the LL

```c
#include<malloc.h>
NODEPTR getnode()
{
    NODEPTR p;
    p=(NODEPTR)malloc(sizeof(NODEPTR));
    return(p);
}

void insert(NODEPTR p,int val)
{
    NODEPTR q;
    q=getnode();
    q->info=val;
    q->next=p->next;
    p->next=q;
}
```
void print(NODEPTR p)
{
    NODEPTR q;
    for(q=p->next;q!=NULL;q=q->next)
    {
        printf("%d \t",q->info);
    }
}
Merging

NODEPTR merge(NODEPTR list1, NODEPTR list2)
{
    if(list1==NULL) return(list2);
    else if(list2==NULL) return(list1);

    else if(list1->info<=list2->info){
        list1->next=merge(list1->next,list2);
        return(list1);
    }

    else{
        list2->next=merge(list1,list2->next);
        return(list2);
    }
}
Pictorial Description

- Dotted line represents initial list. After the recursive calls merge create the solid lines.
### Recursive calls to merge

<table>
<thead>
<tr>
<th>Call</th>
<th>Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>merge(12779,2478)</td>
<td>1224777789</td>
</tr>
<tr>
<td>merge(2779,2478)</td>
<td>22477789</td>
</tr>
<tr>
<td>merge(779,2478)</td>
<td>2477789</td>
</tr>
<tr>
<td>merge(779,478)</td>
<td>477789</td>
</tr>
<tr>
<td>merge(779,78)</td>
<td>77789</td>
</tr>
<tr>
<td>merge(79,78)</td>
<td>7789</td>
</tr>
<tr>
<td>merge(9,78)</td>
<td>789</td>
</tr>
<tr>
<td>merge(9,8)</td>
<td>89</td>
</tr>
<tr>
<td>merge(9,NULL)</td>
<td>9</td>
</tr>
</tbody>
</table>
Splitting the list into 2 equal parts

```c
NODEPTR split(NODEPTR list)
{
    NODEPTR pSecondCell;
    if(list==NULL) return(NULL);
    else if(list->next==NULL) return(NULL);
    else{//list contains more than two elements
        pSecondCell=list->next;
        list->next=pSecondCell->next;
        pSecondCell->next=split(pSecondCell->next);
        return(pSecondCell);
    }
}
```
Pictorial representation of split

Recursive call to split

pSecondCell
The sorting algorithm

NODEPTR MergeSort(NODEPTR list)
{
    NODEPTR SecondList;
    NODEPTR printlist;
    static int i=0;
    i=i+1;
    printlist=getnode();
    printlist->next=list;
    printf("Rec No %d: The curr list is \n",i);
    print(printlist);

    if(list==NULL) return(NULL);
    else if(list->next==NULL) return(list);else{//at least two elements on list
        SecondList=split(list);
        return(merge(MergeSort(list),MergeSort(SecondList)));
    }
}
Snap of the main

NODEPTR list;
list=getnode();
list->next=NULL;
while(val!=-1)
{
    scanf("%d",&val);
    insert(list,val);
}
printf("The entered list is:\n");
print(list);

list->next=.MergeSort(list->next);
printf("\nThe sorted list is:\n");
print(list);
Splitting and Merging Recursively

742897721

72971

4872

791

27

47

82

71

9

4

7

8

2

7

1

122477789

12779

2478

179

27

47

28

17

9

2

7

8

2

7

1
Analyze the merge function

```c
NODEPTR merge(NODEPTR list1, NODEPTR list2) {
    if(list1==NULL) return(list2);                         O(1)+O(1)
    else if(list2==NULL) return(list1);                 O(1)+O(1)
    else if(list1->info<=list2->info){                    O(1)
        list1->next=merge(list1->next,list2);           O(1)+T(n-1)
        return(list1);                                                O(1)
    }
    else{
        list2->next=merge(list1,list2->next);          O(1)+T(n-1)
        return(list2);                                                  O(1)
    }
}
```

\( n \) is the input size of the problem; \( n \) is the sum of the two lists which are merged

\( T(1)=O(1), T(n)=O(1)+T(n-1) \)
RR for merge

- \( T(n) = T(n-1) + O(1) \implies T(n) = O(n) \)
- This is a 1-LiNoReCoCo
- CE has roots, \( r=1 \) with multiplicity 1.
- \( F(n) = O(1) \). Thus the particular solution is \( P(n) = n(C) \)
- Thus \( T(n) = C_1 + nC = O(n) \)
Analyze the split

NODEPTR split(NODEPTR list)
{
    NODEPTR pSecondCell;
    if(list==NULL) return(NULL);          \(O(1)\)
    else if(list->next==NULL) return(NULL); \(O(1)+O(1)\)
    else{\//list contains more than two elements
        pSecondCell=list->next;            \(O(1)\)
        list->next=pSecondCell->next;      \(O(1)\)
        pSecondCell->next=split(pSecondCell->next); \(O(1)+T(n-2)\)
        return(pSecondCell);              \(O(1)\)
    }
}
Analyzing merge sort

NODEPTR MergeSort(NODEPTR list)
{
    if(list==NULL) return(NULL);                                O(1)
    else if(list->next==NULL) return(list);                     O(1)
    else{//at least two elements on list
        SecondList=split(list);                                O(1)+O(n)
        return(merge(MergeSort(list),MergeSort(SecondList)));   2T(n/2)+O(n)
    }
}

n is the number of elements in the list
T(1)=O(1); T(n)=2T(n/2)+O(n)
Solve the RR

- $T(n) = 2T(n/2) + O(n)$
- By Master’s Theorem, $a=2$, $b=2$, $d=1$
- We have $\log_2 2 = 1$
- So, we have $T(n) = O(n \log n)$
- Thus we have a better sorting algorithm in the worst case than the previous sorting algorithms we have seen.
- In fact we have $T(n) = \Theta(n \log n)$
Further Discussions

• We have also discussed on a Divide and Conquer strategy to solve the closest pair problem of n Cartesian points, better than the straight forward $O(n^2)$ algorithm

• Recurrence relation was:
  – $T(n)=2T(n/2)+7n$ which leads to $T(n)=O(n\log n)$