CS 380
ALGORITHM DESIGN AND ANALYSIS

Lecture 3: Merge Sort and Recurrence Equation Analysis

Text Reference: Chapter 2

Bubble Sort

6 5 3 1 8 7 2 4

- Worse case: Comparisons? $O(n^2)$
  Swaps: $O(n^2)$
- Best-case: Comparisons? $\Omega(n)$
  Swaps? $\Omega(1)$
- Average: Comparisons? $\Theta(n^2)$
  Swaps: $\Theta(n^2)$

Another Sorting Algorithm

- What was the running time of insertion sort?
- Can we do better?

Designing Algorithms

- Many ways to design an algorithm:
  - Incremental: *This is what we did with insertion sort. Having sorted the subarray, we insert a single element in its correct position.*
  - Divide and Conquer: *Here the problem is broken up into subproblems that are similar to the original problem but smaller in size. The subproblems are solved recursively then combined to give a solution to the original problem. Merge sort is an example of a divide and conquer algorithm.*

Divide and Conquer

- *Divide the problem into a number of subproblems*
- *Conquer the subproblems by solving them recursively*
- *Combine the subproblem solutions to give a solution to the original problem*

Merge Sort

Merge Sort is an example of a divide and conquer algorithm

```
MERGE-SORT(A, p, r)
//p & r are indices into the array (p < r)
if p < r                    //Check for base case
    q = ⌊(p + r) / 2⌋       //Divide
     MERGE-SORT(A, p, q)        //Conquer M-S
     MERGE-SORT(A, q + 1, r)    //Conquer M-S
     MERGE(A, p, q, r)          //Combine M
```

The Merge Procedure

`MERGE(A, p, q, r)`

- **Input:** Array A and indices p, q, r such that
  - $p \le q < r$
  - Subarray A[p..q] is sorted and subarray A[q+1..r] is sorted. Neither subarray is empty
- **Output:** The two subarrays are merged into a single sorted subarray in A[p..r]

Merge: Example

`MERGE(A, p, q, r)`

index	p=1	2	3	q=4	q+1=5	6	7	r=8
A[index]	1	5	7	9	2	4	6	10

Sorted Sorted

Construct two new arrays L [q – p + 1 + 1] : and R[r – q + 1]:

i	1	2	3	4	5
L[i]	1	5	7	9	inf

j	1	2	3	4	5
R[j]	2	4	6	10	inf

Merge: Example, cont.

i	1	2	3	4	5
L[i]	1	5	7	9	inf

j	1	2	3	4	5
R[j]	2	4	6	10	inf

k	1	2	3	4	5	6	7	8
A[index]	1							

Increment i

i	1	2	3	4	5
L[i]	1	5	7	9	inf

j	1	2	3	4	5
R[i]	2	4	6	10	inf

Merge: Example, cont.

i	1	2	3	4	5
L[i]	1	5	7	9	inf

j	1	2	3	4	5
R[j]	2	4	6	10	inf

k	1	2	3	4	5	6	7	8
A[index]	1	2						

Increment j

i	1	2	3	4	5
L[i]	1	5	7	9	inf

j	1	2	3	4	5
R[i]	2	4	6	10	inf

Merge: Example, cont.

i	1	2	3	4	5
L[i]	1	5	7	9	inf

j	1	2	3	4	5
R[j]	2	4	6	10	inf

k	1	2	3	4	5	6	7	8
A[index]	1	2	4					

Increment j

i	1	2	3	4	5
L[i]	1	5	7	9	inf

j	1	2	3	4	5
R[i]	2	4	6	10	inf

Merge: Example, cont.

i	1	2	3	4	5
L[i]	1	5	7	9	inf

j	1	2	3	4	5
R[j]	2	4	6	10	inf

k	1	2	3	4	5	6	7	8
A[index]	1	2	4	5	6	7	9	

Increment i

i	1	2	3	4	5
L[i]	1	5	7	9	inf

j	1	2	3	4	5
R[i]	2	4	6	10	inf

Merge: Example, cont.

i	1	2	3	4	5
L[i]	1	5	7	9	inf

j	1	2	3	4	5
R[j]	2	4	6	10	inf

k	1	2	3	4	5	6	7	8
A[index]	1	2	4	5	6	7	9	10

Increment j

i	1	2	3	4	5
L[i]	1	5	7	9	inf

j	1	2	3	4	5
R[i]	2	4	6	10	inf

Done

The Merge Algorithm: Cost?

MERGE(A,p,q,r)

1	n1 = q - p +1
2	n2 = r - q
3	let L[1..n_1+1] and R[1..n_2+1] be new arrays
4	for i = 1 to n_1
5	L[i] = A[p + i -1]
6	for j = 1 to n_2
7	R[j] = A[q + j]
8	L[n_1 + 1] = infinity // Not necessary if careful
9	R[n_2 + 1] = infinity // Not necessary if careful
10	i = 1
11	j = 1
12	for k = p to r
13	if L[i] <= R[j]
14	A[k] = L[i]
15	i = i + 1
16	else A[k]= R[j]
17	j = j + 1

Merge Sort

Merge Sort is an example of a divide and conquer algorithm

```
MERGE-SORT(A, p, r)
//p & r are indices into the array (p < r)
if p < r                    //Check for base case
    q = ⌊(p + r) / 2⌋       //Divide
     MERGE-SORT(A, p, q)        //Conquer M-S
     MERGE-SORT(A, q + 1, r)    //Conquer M-S
     MERGE(A, p, q, r)          //Combine M
```

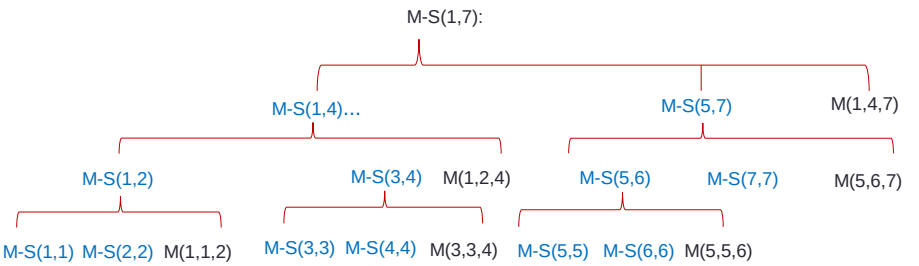

Merge Sort: Example

Each function call `MERGE-SORT(A, p, r)` results in three function calls:

```
q = ⌊(p + r) / 2⌋ // Get midpoint
MERGE-SORT(A, p, q)  == M-S(p,q)
MERGE-SORT(A, q + 1, r)== M-S(q+1,r)
MERGE(A, p, q, r)== M(p,q,r)
```

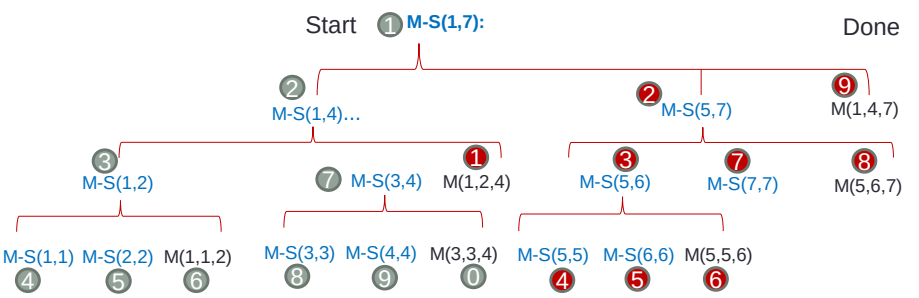
Best illustrated with a tree:

Merge Sort: Calls, A[7], p = 1, r = 7



$q = \lfloor (p + r) / 2 \rfloor$

Merge Sort: Calls, A[7], p = 1, r = 7



Order: 1 …. 9 0 1 2 .. 9

Left Right

$q = \lfloor (p + r) / 2 \rfloor$

Merge Sort: Example, p = 1, r = 7

Index (j)	1	2	3	4	5	6	7
Value A[j]	3	1	7	4	8	2	6

Recurrence Relation

- Let T(n) be the time for Merge-Sort to execute on an n element array.
- The time to execute on a one element array is O(1)
- Then we have the following relationship:

$$T(n) = 2\,T(n/2) + O(n) \qquad \text{[the O(n) is for Merge]}$$
$$T(1) = O(1)$$

Merge Sort

- To solve the recurrence relation we'll write n instead of O(n) as it makes the algebra simpler:

$$T(n) = 2\,T(n/2) + n$$

$$T(1) = 1$$

- Solve the recurrence by iteration (substitution)

Merge Sort

$$\begin{aligned} T(n) &= 2\,T(n/2) + n \\ &= 2\,[\mathbf{2\,T(n/4) + n/2}] + n \\ &= 4\,T(n/4) + 2n \\ &= 4\,[\mathbf{2\,T(n/8) + n/4}] + 2n \\ &= 8\,T(n/8) + 3n \\ &= \\ &\ldots. \\ &= \end{aligned}$$

General Formula?

Derivation will end when we get to T(1)=1. What does this imply about *n = f(k)?*

Merge Sort

Continuing, we get:

$T(n) = \ldots$
$= 2^k T(n/2^k) + k\,n$
$= 2^{\log_2 n} T(1) + (\log_2 n)\, n$
$= n + n \log_2 n$ [*remember that T(1) = 1*]
$= O(n \log n)$

Insertion sort

$T(n) = T(n-1)+(n-1)$, $T(1) = 1$

Challenges

$T(n) = T(n-1) + 2n - 1, \quad T(0) = 0$

Tower of Hanoi:

$T(n)=2T(n-1) + 1, T(0)=1$

Next Time

- Alternative analysis of Merge Sort
- Master Method for Solving Recurrences.