Quicksort and Mergesort

This week

- ❑Quicksort algorithm
- ❑Quicksort performance
- ❑Quicksort analysis
- ❑Mergesort algorithm
- ❑Mergesort analysis

Further Reading
Chapters 1 and 8 from Textbook

Quicksort

- **The worst case running time of Quicksort algorithm is $O(n^2)$**
- **However, its expected running time is O (n log n)**
- **Three-step divide-and-conquer process for sorting a subarray A[*l*..r]**

Divide : partition the array A[*l*..r] into two nonempty subarrays A[*l*..q] and A[q+1,r] such that each element of A[*l*..q] is less than or equal to each element of A[q+1,..,r]

Conquer : sort the two subarrays A[*l*..q] and A[q+1..r] by recursive calls to Quicksort

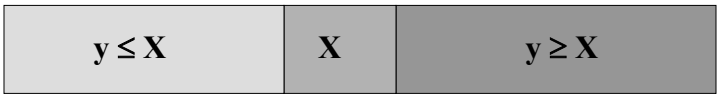
Combine : the subarrays are already sorted in place. No work is needed to combine them

Example

13	02	18	26	76	87	98	11	93	77	65	43	38	09	65	06
13	02	06	26	76	87	98	11	93	77	65	43	38	09	65	18
13	02	06	09	76	87	98	11	93	77	65	43	38	26	65	18
13	02	06	09	11	87	98	76	93	77	65	43	38	26	65	18
11	02	06	09	13	87	98	76	93	77	65	43	38	26	65	18
11	02	06	09	13	87	98	76	93	77	65	43	38	26	65	18
09	02	06	11	13	87	98	76	93	77	65	43	38	26	65	18
09	02	06	11	13	87	98	76	93	77	65	43	38	26	65	18
06	02	09	11	13	87	98	76	93	77	65	43	38	26	65	18
06	02	09	11	13	87	98	76	93	77	65	43	38	26	65	18
02	06	09	11	13	87	98	76	93	77	65	43	38	26	65	18
02	06	09	11	13	87	18	76	93	77	65	43	38	26	65	98
02	06	09	11	13	87	18	76	65	77	65	43	38	26	93	98
02	06	09	11	13	26	18	76	65	77	65	43	38	87	93	98
02	06	09	11	13	18	26	76	65	77	65	43	38	87	93	98
02	06	09	11	13	18	26	76	65	38	65	43	77	87	93	98
02	06	09	11	13	18	26	43	65	38	65	76	77	87	93	98
02	06	09	11	13	18	26	43	38	65	65	76	77	87	93	98
02	06	09	11	13	18	26	38	43	65	65	76	77	87	93	98
02	06	09	11	13	18	26	38	43	65	65	76	77	87	93	98

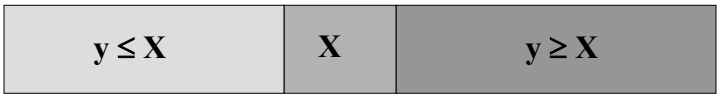
Procedure **Quicksort(A,*l*,r)**
Input : Unsorted Array (A,*l*,r)
Output : Sorted subarray A(0..r)
To sort the entire array A, ***l* = 1** and **r = length[A]**

```
if l < r
    then q ← PARTITION(A,l,r)
         QUICKSORT(A,l,q-1)
         QUICKSORT(A,q+1,r)
```



Procedure **PARTITION(A,*l*,r)**
Input : Array A(*l* .. r)
Output : A and q such that A[i] ≤ A[q] for all i ≤ q and
A[j] > A[q] for all j > q.

```
x ← A[l]; i ← l; j ←r;
while i < j do
    while A[i] ≤ x and i ≤ r do i ← i +1;
    while A[j] > x and j ≥ l do j ← j -1;
    if i< j then
        exchange A[i] ↔ A[j];
q ← j;
exchange A[l] ↔ A[q];
```



Running Time of Quicksort

T(n) = n-1 + T(i-1) + T(n-i)

It takes n-1 comparisons for the partition
Then we sort smaller sequences of size i-1 and n-i
Each element has the same probability of being selected as the pivot,
The average running time is given by,

$$T(n) = n - 1 + \frac{1}{n}\sum_{i=1}^{n} T(i-1) + \frac{1}{n}\sum_{i=1}^{n} T(n-i)$$

$$T(n) = n - 1 + \frac{2}{n}\sum_{i=1}^{n} T(i) = O(n \log n)$$

Detailed Analysis in the Class

Mergesort

Like Quicksort, Mergesort algorithm also is based on the divide-and–conquer principle.

Divide: This step computes the middle of the array, takes constant time, Θ(1)

Conquer : Two subproblems, each of size n/2 are recursively solved.
Each subproblem contributes 2T(n/2) to the running time.

Combine: Two sorted sequences are merged, this takes Θ(n) time

Example

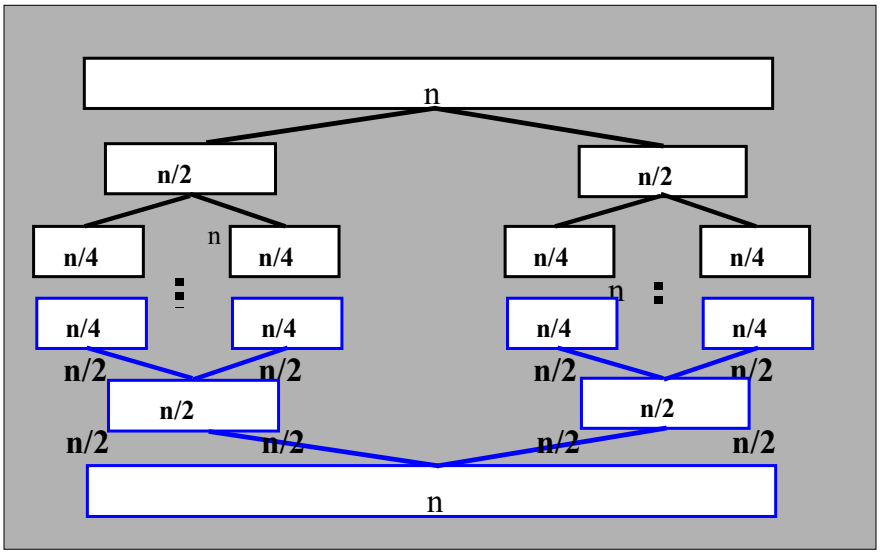
13	02	18	26	76	87	98	11	93	77	65	43	38	09	65	06
13	02	18	26	76	87	98	11	93	77	65	43	38	09	65	06
13	02	18	26	76	87	98	11	93	77	65	43	38	09	65	06
02	13	18	26	76	87	98	11	93	77	65	43	38	09	65	06
02	13	18	26	76	87	98	11	93	77	65	43	38	09	65	06
02	13	18	26	76	87	98	11	93	77	65	43	38	09	65	06
02	13	18	26	76	87	98	11	93	77	65	43	38	09	65	06
02	13	18	26	76	87	98	11	93	77	65	43	38	09	65	06
02	13	18	26	76	87	11	98	93	77	65	43	38	09	65	06
02	13	18	26	11	76	87	98	93	77	65	43	38	09	65	06
02	11	13	18	26	76	87	98	93	77	65	43	38	09	65	06
02	11	13	18	26	76	87	98	06	09	38	43	65	65	77	99
02	06	09	11	13	18	26	38	43	65	65	76	77	87	93	98

```
Procedure MERGESORT(A,l,r)
Input : A an array in the range 1 to n.
Output: Sorted array A.
        if l < r
                then q ← ⌊(l+r )/2⌋ ;
                        MERGESORT(A,l,q)
                        MERGESORT(A,q+1, r)
                        MERGE (A,l,q,r)
```



```
Procedure MERGE(A,l,q,r)
 Inputs: two sorted subarrays A(l, q) and A(q+1, r)
 Output : Merged and sorted array A(l, r)
        i ← l;
        j ← q+1;
        k ← 0;
        while (i ≤ q) and ( j ≤ r) do
                k  ← k+1;
                if A[i] ≤ A[j] then
                        TEMP[k] ← A[i];
                        i ← i +1;
                else
                        TEMP[k] ← A[j];
                        j ← j +1;
        if j > r  then
                for t ←0 to q – i do
                        A[r-t] ← A[q-t];
        for t ← 0 to k-1 do
                A[l +t] ← TEMP[t];
```

Runtime Complexity of Mergesort



Runtime Complexity of Mergesort

$T(n) = 2\,T(n/2) + \Theta(n)$
$T(n/2) = 2\,T(n/4) + n/2$
$T(n/4) = 2\,T(n/8) + n/4$
$T(n) = 2\{2\,T(n/4) + n/2\} + n = 4\,T(n/4) + 2\,n$
$T(n) = 2^3\,T(n/2^3) + 3n$
. . .
$T(n) = 2^k\,T(n/2^k) + k\,n$ If $n = 2^k$ then , $k = \log n$

Therefore, $T(n) = 2^k\,T(1) + n \log n$
$= n\,\Theta(1) + n \log n$
$T(n) = O(n \log n)$

Questions

- What is a pivot element?
- Do you understand divide-and-conquer?
- What is running time if pivot element is at the center of the array?
- How is Mergesort different from Quicksort?
- Trace the algorithm with the help of an example?
- What is the use of TEMP array?
- Do you know why we execute steps 13 and 14 in the MERGE algorithm?
- What happens if i >q after the while loop (4-11)?

- Find applications for selection sort, heapsort, quicksort, and mergesort sorting algorithms?
- Which of these problems are suitable for different types of sorting operations?
- Identify applications for which each sorting algorithm works best

- Insertion Sort
- Counting Sort
- Find the Maximum
- Find the Minimum

The Odd-Even Mergesort problem

Porcedure PARALLEL ODD-EVEN MERGE SORT

```
1.      for  i ←1 to ⌈N/2⌉; N = number of data elements
2.          for  all PE_j such that j is odd  and 1≤ j ≤ N-1
3.              if [PE_j].a  > [PE_{j+1}].a
4.                      swap([PE_j].a,[PE_{j+1}].a)
5.          for all PE_j such that j is even and 1≤ j ≤ N-1
6.              if [PE_j].a  > [PE_{j+1}].a
7.                      swap([PE_j].a,[PE_{j+1}].a)
}
```

9 18 27 36 45 54 63 72 64 56 48 40 32 24 16 8

Bitonic Mergesort

A bitonic sequence is a sequence of numbers,

$a_0, a_1, \ldots, a_{n-1}$ such that for $0 \le i \le n-1$

$a_0, a_1, \ldots, a_i$ is in ascending order and

$a_{i+1}, a_{i+2}, \ldots, a_{n-1}$ is in descending order

let, $n = 2^k$, and $i = n/2 - 1$.

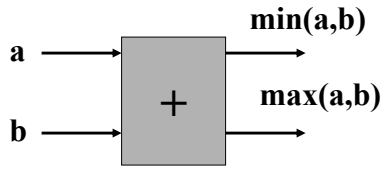
$a_0 \le a_1 \le a_2, \ldots, a_{n/2-2} \le a_{n/2-1}$ . . . and

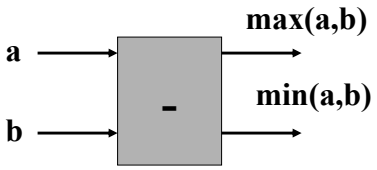
$a_{n/2} \ge a_{n/2+1} \ge a_{n/2+2} \ge a_{n/2+3} \ldots a_{n-2} \ge a_{n-1}$

Consider, $\min(a_0, a_{n/2})$, $\min(a_1, a_{n/2+1})$, . . ., $\min(a_{n/2-1}, a_{n-1})$
$\max(a_0, a_{n/2})$, $\max(a_1, a_{n/2+1})$, . . . , $\max(a_{n/2-1}, a_{n-1})$

Both are bitonic sequences!!

Sorting Networks



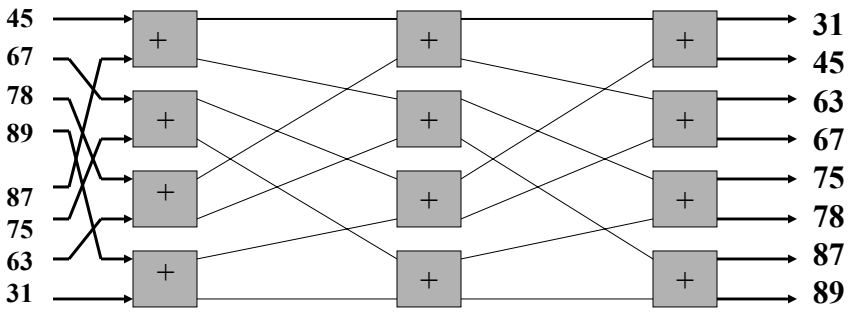


compare_exchange_min(j); compare_exchange_max(j);

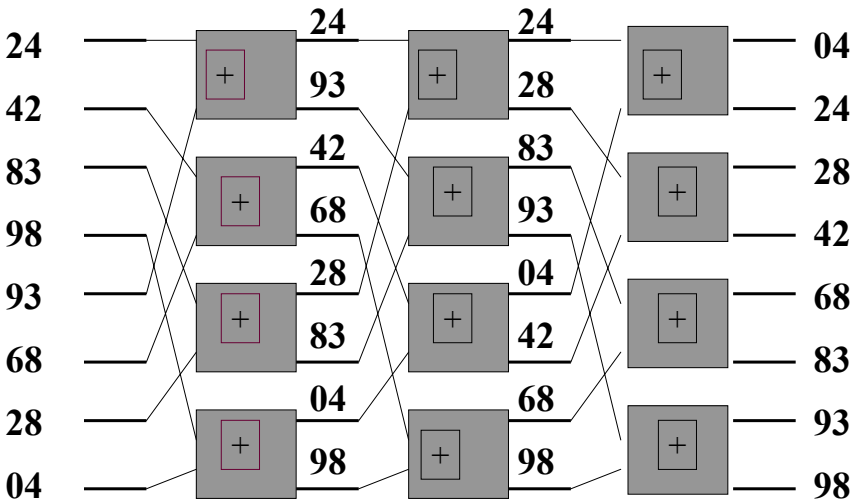
Compare a_0 with $a_{n/2}$
a_1 with $a_{n/2+1}$
.
.
.
$a_{n/2-1}$ with a_{n-1}

Compare data elements i and $2^{k-1}+i$ and put the min in position 2i and the max in position 2i+1

Shuffle-Exchange Network Week 6



n/2 switches per stage
log n stages



Parallel Bitonic Sort Algorithm

1. Compare [PE(i)].a with [PE(i+2^{k-1})].a
   PE[2i] ← MIN { [PE(i)].a, [PE(i+2^{k-1})].a}
   PE[2i+1] ← MAX {[PE(i)].a, [PE(i+2^{k-1})].a}
2. Compare [PE(i)].a with [PE(i+2^{k-1})].a
   PE[2i] ← MIN { [PE(i)].a, [PE(i+2^{k-1})].a}
   PE[2i+1] ← MAX {[PE(i)].a, [PE(i+2^{k-2})].a}

. . .

k. Compare [PE(i)].a with [PE(i+2^{k-1})].a and
   PE[2i] ← MIN { [PE(i)].a, [PE(i+2^{k-1})].a}
   PE[2i+1] ← MAX {[PE(i)].a, [PE(i+2^{k-2})].a}

for j = 1 to k
   Compare [PE(i)].a with [PE(i+2^{k-1})].a
   PE[2i] ← MIN { [PE(i)].a, [PE(i+2^{k-1})].a}
   PE[2i+1] ← MAX {[PE(i)].a, [PE(i+2^{k-1})].a}

Given Unsorted list :

98 24 42 83 68 04 93 28

98 24 42 and 83 in ascending order and

68 04 93 and 28 in descending order

98 24 42 83	68 04 93 28
98 24 42 83	68 04 93 28
24 98 83 42	68 04 28 93
24 42 83 98	68 93 28 04
24 42 83 98	93 68 28 04

98 24 42 83 68 04 93 28