

Data Structures:

Internal Sorting: Insertion Sort, Selection Sort, Merge Sort, Quick Sort

YoungWoon Cha

(Slide credits to Won Kim)

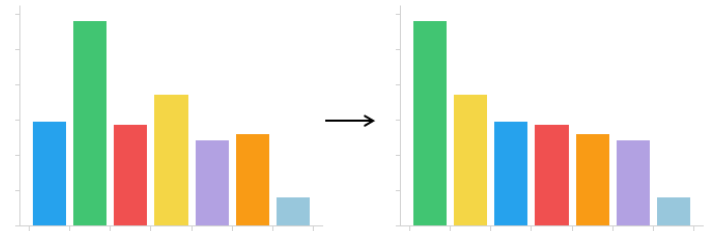
Spring 2022



Sorting - Internal

Sorting

- Definition
 - Arranging elements of a list in some order
 - Lexicographical order (symbols, numbers)
 - Ascending order or descending order
 - Composite-key sort
- Why sort?
 - Required end result
 - Easy visual search
 - Efficient intermediate data structure for computing the end result
- In general, entire records are sorted and change positions.





Motivating Example 1

name	age	salary	employer
Kim	25	200	LG
Lee	30	300	IBM
Park	23	250	LG
Cho	40	500	Samsung
...			
Chung	28	900	Samsung



Motivating Example 1 (cont'd)

name	age	salary	employer
Cho	40	500	Samsung
Chung	28	900	Samsung
Kim	25	200	LG
Lee	30	300	IBM
...			
Park	23	250	LG

**This is easier
to search
by name**



Motivating Example 2

name	age	salary	employer
Kim	25	200	LG
Lee	30	300	IBM
Park	23	250	LG
Cho	40	500	Samsung
...			
Chung	28	900	Samsung

**Who works for
Samsung?**

**Who is the
oldest?
highest paid?**



Motivating Example 2 (cont'd)

name	age	salary	employer
Chung	28	900	Samsung
Cho	40	500	Samsung
Park	23	250	LG
Kim	25	200	LG
...			
Lee	30	300	IBM

**Who works for
Samsung?**

**Who is the
oldest?
highest paid?**



What the Sorted Results Make Possible

- Speedy sequential search (that can stop in the middle of a search)
- Binary search ($O(\log_2 n)$ performance)
- Fast join of multiple tables (in a relational database)



Join: What is the number of employees of the company where Cho works?

To find the answer, we need to join two tables: employee and company.

employee

name	age	salary	employer
Kim	25	200	LG
Lee	30	300	IBM
Park	23	250	LG
Cho	40	500	Samsung
...			
Chung	28	900	Samsung

company

company	years	country	employees
IBM	107	USA	450,000
LG	60	Korea	38,000
Samsung	49	250	310,000
SAP	46	500	15,000



Sorting Algorithms

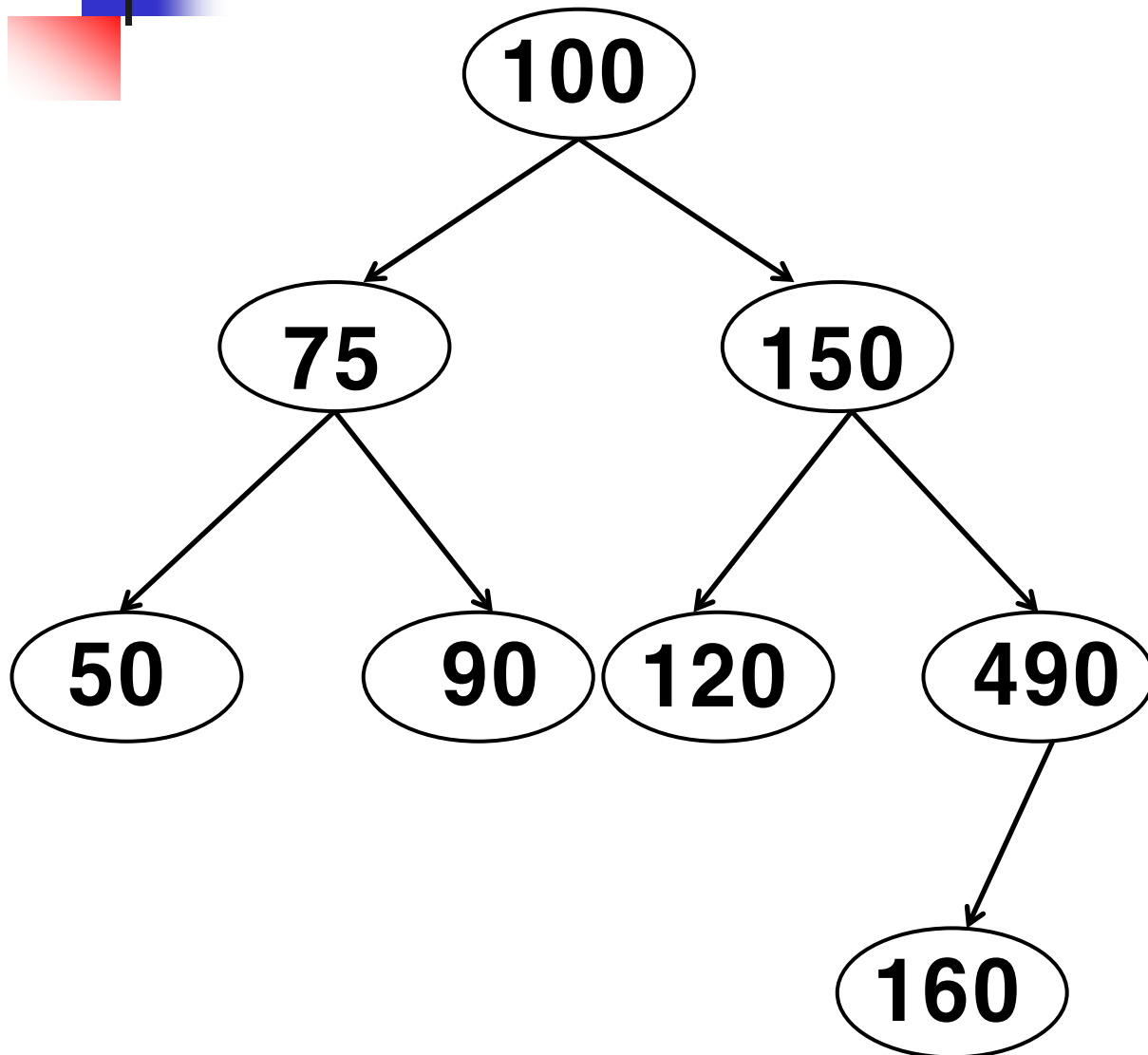
- Internal Sorting
 - List to be sorted fits in main memory.
- External Sorting
 - List to be sorted resides on secondary storage.
- Time Complexity & Space Complexity
- Stable Sorting
 - "equal" elements are ordered in the same order in the sorted list.
 - The relative order of the duplicate elements should not change.
- In-place Sorting
 - A sort algorithm in which the sorted items occupy the same storage as the original ones.
 - These algorithms may use $O(n)$ additional memory for bookkeeping, but at most a constant number of items are kept in auxiliary memory at any time. $O(1)$



Sorting Algorithms

- Internal Sorting
 - Insertion sort
 - Selection sort
 - (Bubble sort – worst; please self-study it.)
 - Merge sort
 - Quick sort
 - Heap sort
 - Radix sort

Inorder Traversal of a Binary Search Tree: Outputs Keys in Ascending **Sort Order**



50
75
90
100
120
150
160
490



Insertion Sort



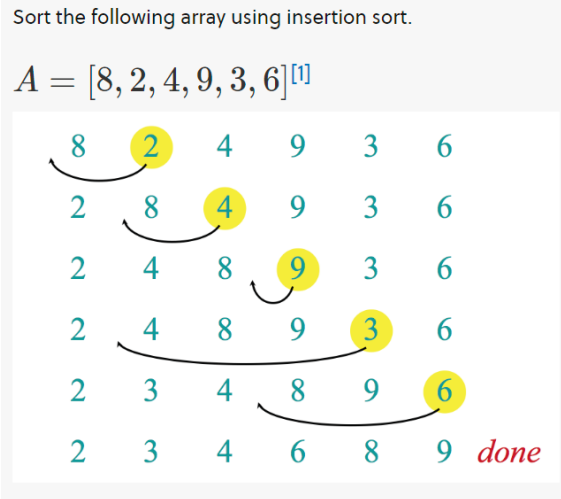
Insertion Sort: Concept

- Start with a sorted list.
- To insert a new key, search the list for a correct position, and insert it there.

insert: 170	(170)
insert: 90	(90, 170)
insert: 2	(2, 90, 170)
insert: 802	(2, 90, 170, 802)
insert: 24	(2, 24, 90, 170, 802)
insert: 45	(2, 24, 45, 90, 170, 802)
insert: 75	(2, 24, 45, 75, 90, 170, 802)
insert: 66	(2, 24, 45, 66, 75, 90, 170, 802)

Insertion Sort: Method

- Unsorted Array → Sorted Array
 - Note the SWAP operation!



- Examples:
 - <https://youtu.be/OGzPmgsI-pQ>
 - <https://www.hackerearth.com/practice/algorithms/sorting/insertion-sort/visualize/>



Insertion Sort: Exercise

sort the list:

22, 25, 7, 3, 30, 11, 14, 8

sort the list:

10, 20, 30, 40, 50, 60

sort the list:

60, 50, 40, 30, 20, 10



Insertion Sort: Exercise

sort the list:

22, 25, 7, 3, 30, 11, 14, 8

insert: 22	(22)
insert: 25	(22, 25)
insert: 7	(7, 22, 25)
insert: 3	(3, 7, 22, 25)
insert: 30	(3, 7, 22, 25, 30)
insert: 11	(3, 7, 11, 22, 25, 30)
insert: 14	(3, 7, 11, 14, 22, 25, 30)
insert: 8	(3, 7, 8, 11, 14, 22, 25, 30)



Insertion Sort: Performance and Qualities

- Average & Worst Case: $O(n^2)$
 - n = the number of keys in the list
 - $O(n)$: time to swap the keys for the correct position for a key on the current sorted list
- Best Case: $O(n)$
 - If the list is already sorted, the swap becomes $O(1)$
- Space Complexity: $O(1)$ (in-place sort)
- Simple and good for a short list
- Good when the list is already partially sorted.
- Stable Sorting

10, 10, 10, 20, 30, 40



Insertion Sort: Example code

```
// C program for insertion sort
#include <math.h>
#include <stdio.h>

/* Function to sort an array using insertion sort*/
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

        /* Move elements of arr[0..i-1], that are
        greater than key, to one position ahead
        of their current position */
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

// A utility function to print an array of size n
void printArray(int arr[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

/* Driver program to test insertion sort */
int main()
{
    int arr[] = { 12, 11, 13, 5, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);

    insertionSort(arr, n);
    printArray(arr, n);

    return 0;
}
```

<https://www.geeksforgeeks.org/insertion-sort/>



Selection Sort



Selection Sort

- $n-1$ passes over a list of n keys
- On the i^{th} pass
 - select the largest key among the first $n-i$ keys, and exchange it with the $n-i+1^{\text{th}}$ key
- Reading
 - http://en.wikipedia.org/wiki/Selection_sort



Selection Sort: Example

sort the list:

13, 4, 9, 21, 37, 17, 22, 3, 8

pass 1: **13, 4, 9, 21, 8, 17, 22, 3, 37**

pass 2: **13, 4, 9, 21, 8, 17, 3, 22, 37**

pass 3: **13, 4, 9, 3, 8, 17, 21, 22, 37**

.....

pass 8: **3, 4, 8, 9, 13, 17, 21, 22, 37**



Selection Sort: Exercise

sort the list:

22, 25, 7, 3, 30, 11, 14, 8

Pass 1: 22, 25, 7, 3, 8, 11, 14, 30

Pass 2: 22, 14, 7, 3, 8, 11, 25, 30

Pass 3: 11, 14, 7, 3, 8, 22, 25, 30

Pass 4: 11, 8, 7, 3, 14, 22, 25, 30

Pass 5: 3, 8, 7, 11, 14, 22, 25, 30

Pass 6: 3, 7, 8, 11, 14, 22, 25, 30

Pass 7: 3, 7, 8, 11, 14, 22, 25, 30



Selection Sort: Method

- On the i^{th} pass
 - It's fine to select the smallest key.
 - Note how the SWAP operation works.
- Examples:
 - <https://www.youtube.com/watch?v=xWBP4IzkoyM>
 - <https://youtu.be/wnKQsow7ERl>
 - <https://www.hackerearth.com/practice/algorithms/sorting/insertion-sort/visualize/>



Selection Sort: Exercise

sort the list:

22, 25, 7, 3, 30, 11, 14, 8



Selection Sort: Performance

- $O(n^2)$ performance (Best, Average, Worst)
 - n is the number of keys in the list
 - n^2 comparisons
 - generally worse than insertion sort
 - $O(n)$ exchanges
- Space Complexity: $O(1)$ (in-place sort)
- Useful for a small list stored in EEPROM or Flash.
 - “exchanges” require writing to an array.
 - For an array stored in EEPROM or Flash, writing to memory is significantly more expensive than reading.
- Unstable sorting

20, 30, 20, 10, 40, 50



Selection Sort: Example code

```
// C program for implementation of selection sort
#include <stdio.h>

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    // One by one move boundary of unsorted subarray
    for (i = 0; i < n-1; i++)
    {
        // Find the minimum element in unsorted array
        min_idx = i;
        for (j = i+1; j < n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;

        // Swap the found minimum element with the first element
        swap(&arr[min_idx], &arr[i]);
    }
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver program to test above functions
int main()
{
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr)/sizeof(arr[0]);
    selectionSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

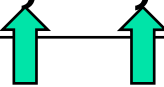
<https://www.geeksforgeeks.org/selection-sort/>



Duplex Selection Sort

- $(n-1)/2$ passes over a list of n keys
 - Select the largest key and the smallest key, and
 - Exchange the largest key with the last key, and
 - Exchange the smallest key with the first key.

22, 25, 7, 3, 30, 11, 14, 8



**For each pass,
select the maximum & minimum at the same time**

- $O(n^2)$ performance
 - n is the number of keys in the list



Merge Sort



Merge Sort

- Divide and Conquer
 - **Divide** the problem into multiple small problems
 - **Conquer** the subproblems
 - **Combine** the solutions of the subproblems
- Split and Merge
 - Split: Divide a list successively into two **sorted** sub-lists.
 - Merge: Merge the two sub-lists successively into a single sorted list.
- Invented by John von Neumann in 1945
- Supplementary Reading
 - http://en.wikipedia.org/wiki/Merge_sort



Merge Example

merge two sorted lists:

15, 20, 25, 30, 35



7

15, 20, 25, 30, 35



7, 15

15, 20, 25, 30, 35



7, 15, 18

7, 18, 22, 28, 34



7, 18, 22, 28, 34



7, 18, 22, 28, 34





Merge Example (cont'd)

15, 20, 25, 30, 35



7, 15, 18, 20

15, 20, 25, 30, 35



7, 15, 18, 20, 22

15, 20, 25, 30, 35



7, 15, 18, 20, 22, 25

7, 18, 22, 28, 34



7, 18, 22, 28, 34



7, 18, 22, 28, 34





Merge Example (cont'd)

15, 20, 25, 30, 35

7, 18, 22, 28, 34

↑
7, 15, 18, 20, 22, 25, 28

15, 20, 25, 30, 35

7, 18, 22, 28, 34

↑
7, 15, 18, 20, 22, 25, 28, 30

15, 20, 25, 30, 35

7, 18, 22, 28, 34

↑
7, 15, 18, 20, 22, 25, 28, 30, 34, 35



Exercise: Merge

merge two sorted lists:

3, 9, 20, 25, 35, 40

7, 18, 22, 28, 34



Exercise: Merge

merge two sorted lists:

3, 9, 20, 25, 35, 40



3

3, 9, 20, 25, 35, 40



3, 7

3, 9, 20, 25, 35, 40



3, 7, 9

7, 18, 22, 28, 34



7, 18, 22, 28, 34



7, 18, 22, 28, 34





Exercise: Merge

merge two sorted lists:

3, 9, 20, 25, 35, 40



3, 7, 9, 18

3, 9, 20, 25, 35, 40



3, 7, 9, 18, 20

3, 9, 20, 25, 35, 40



3, 7, 9, 18, 20,
22

7, 18, 22, 28, 34



7, 18, 22, 28, 34



7, 18, 22, 28, 34





Exercise: Merge

merge two sorted lists:

3, 9, 20, 25, 35, 40



3, 7, 9, 18, 20, 22,

25

3, 9, 20, 25, 35, 40



3, 7, 9, 18, 20, 22, 25,

28

3, 9, 20, 25, 35, 40



3, 7, 9, 18, 20, 22, 25, 28,

34

7, 18, 22, 28, 34



7, 18, 22, 28, 34



7, 18, 22, 28, 34





Exercise: Merge

merge two sorted lists:

3, 9, 20, 25, 35, 40

7, 18, 22, 28, 34



3, 7, 9, 18, 20, 22, 25, 28,
34, 35

3, 9, 20, 25, 35, 40

7, 18, 22, 28, 34



3, 7, 9, 18, 20, 22, 25, 28, 34,
35, 40



Merge Sort

sort the list:

22, 25, 7, 3, 30, 11, 14, 8

splitting phase

22, 25, 7, 3

30, 11, 14, 8

22, 25

7, 3

30, 11

14, 8

22

25

7

3

30

11

14

8



Merge Sort: (cont'd)

merging phase



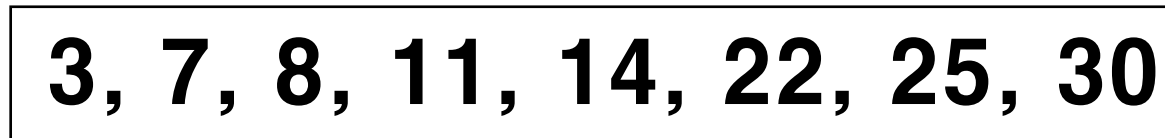
pass 1:



pass 2:

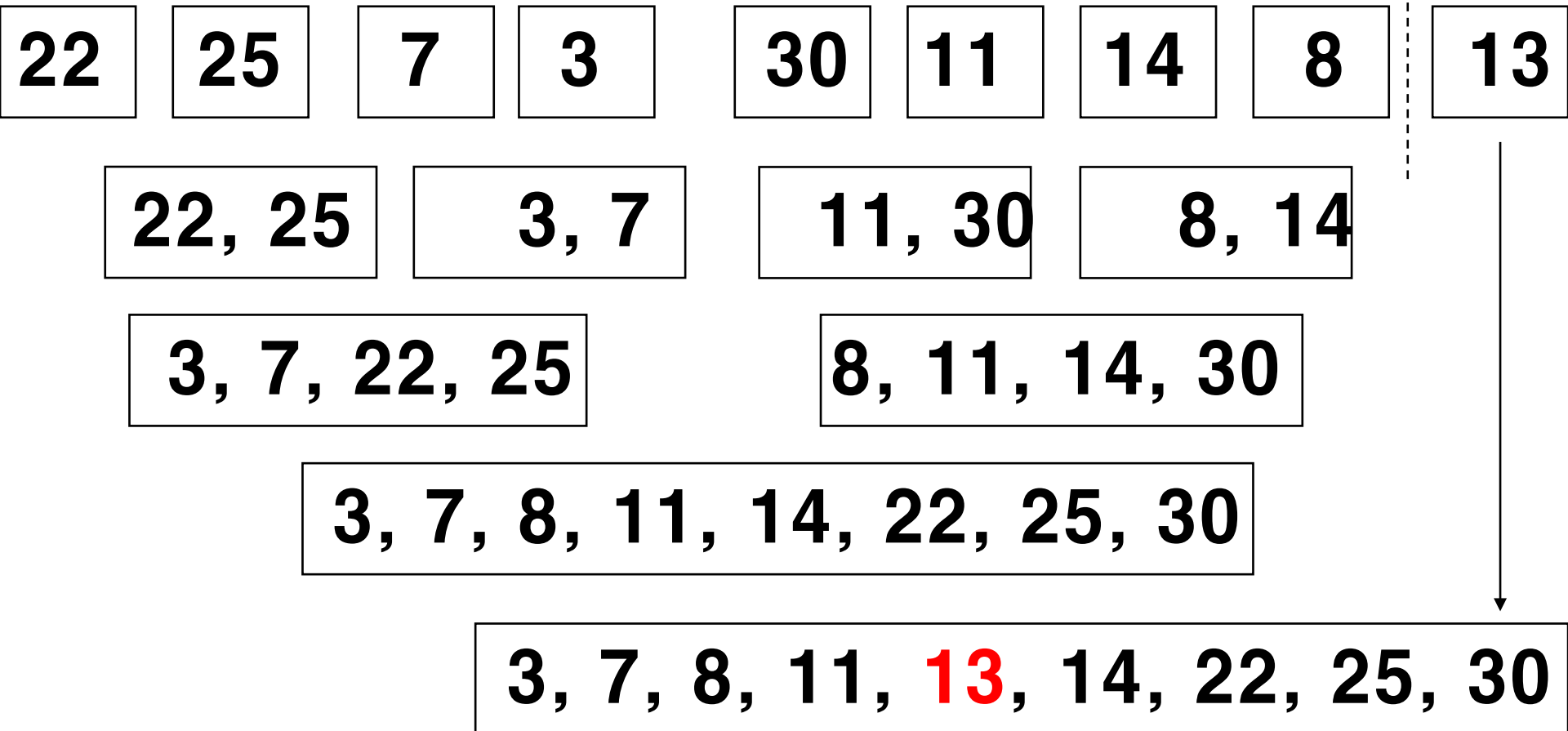


pass 3:



Merge Sort: Merging an Odd Number of Data

merging phase





Merge Sort: Performance and Qualities

- Merge: $O(n)$
 - The cursors always move to the right since the sublists are already sorted.
- $O(n \log_2 n)$ comparisons and copying of keys
 - n : number of keys in the list
 - $\log_2 n$ times Merge (guaranteed) (Best, Average, Worst)
 - best “worst case” sorting
- Space Complexity: $O(n)$ (**Not** in-place sort)
 - The sublists requires equal amount of additional space as the unsorted array.
- Stable Sort
 - Since Merge does not swap for duplicates.
- Amenable to parallel processing
- Amenable to adaptation for external sorting

Merge Sort: Example code

```
/* C program for Merge Sort */
#include <stdio.h>
#include <stdlib.h>

// Merges two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    /* create temp arrays */
    int L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    /* Merge the temp arrays back into arr[l..r] */
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    /* Copy the remaining elements of L[], if there
    are any */
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    /* Copy the remaining elements of R[], if there
    are any */
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}
```

```
/* l is for left index and r is right index of the
sub-array of arr to be sorted */
void mergeSort(int arr[], int l, int r)
{
    if (l < r) {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l + (r - l) / 2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);
    }
}

/* UTILITY FUNCTIONS */
/* Function to print an array */
void printArray(int A[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
}

/* Driver code */
int main()
{
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    printf("\nSorted array is \n");
    printArray(arr, arr_size);
    return 0;
}
```



Performance Comparison



N^2 vs. $N \log_2 N$

N^2

	$n = 10^3$	$n = 10^6$	$n = 10^9$
PC		2.8 hr	317 yr
super com		1 sec	1.7 wk

$N \log_2 N$

	$n = 10^3$	$n = 10^6$	$n = 10^9$
PC		1 sec	18 min
super com			

pc: 10^8 compares/sec

supercom: 10^{12} compares/sec



Quick Sort



Quick Sort

- One of the most widely used sorting algorithms
- Sort by divide and conquer
- Invented by C.A.R. Hoare in 1961
- Types of quick sort
 - Plain quick sort (creates new lists for intermediate results)
 - In-place quick sort (uses the original list)
- Supplementary Reading
 - <http://en.wikipedia.org/wiki/Quicksort>
 - <http://www.cs.purdue.edu/homes/ayg/CS251/slides/chap8b.pdf>



Quick Sort

- Select a Pivot.
 - Arbitrary item on the list (first, last, middle,...)
- Partition
 - Move Keys $<$ Pivot into a L-List.
 - Move Keys $=$ Pivot into a P-List.
 - Move Keys $>$ Pivot into a R-List.
- Recurse on the L-List, and R-List, until both lists are sorted.
- Concatenate L-List, P-List, R-List into a new list.



Quick Sort Pseudo Code

QUICKSORT(A, p, r)

```
1  if  $p < r$ 
2       $q = \text{PARTITION}(A, p, r)$ 
3      QUICKSORT( $A, p, q - 1$ )
4      QUICKSORT( $A, q + 1, r$ )
```

PARTITION(A, p, r)

```
1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $A[r]$ 
8  return  $i + 1$ 
```



Quick Sort: Example 1

sort the list: (pivot=first element)

70, 90, 12, 80, 24, 45, 75, 66

pass 1: 12, 24, 45, 66, 70, 90, 80, 75

pass 2: 12, 24, 45, 66, 70, 80, 75, 90

pass 3: 12, 24, 45, 66, 70, 75, 80, 90

pass 4: 12, 24, 45, 66, 70, 75, 80, 90



Quick Sort: Exercise

sort the list:(pivot=first element)
17, 90, 2, 80, 14, 13, 75, 16



Quick Sort: Solution

sort the list:(pivot=first element)

17, 90, 2, 80, 14, 13, 75, 16

pass 1: 2, 14, 13, 16, 17, 90, 80, 75

pass 2: 2, 14, 13, 16, 17, 80, 75, 90

pass 3: 2, 13, 14, 16, 17, 75, 80, 90



Quick Sort: Exercise

sort the list:(pivot=last element)

17, 90, 2, 80, 14, 13, 75, 16



Quick Sort: Exercise

sort the list:(pivot=last element)

17, 90, 2, 80, 14, 13, 75, 16

pass 1: 2, 14, 13, 16, 17, 90, 80, 75

pass 2: 2, 13, 14, 16, 17, 75, 90, 80

pass 3: 2, 13, 14, 16, 17, 75, 80, 90



Quick Sort: Example 2 (**worst case**)

sort the list: (pivot=first element)

2, 24, 45, 66, 75, 90, 170, 802

pass 1: **2, 24, 45, 66, 75, 90, 170, 802**
pass 2: **2, 24, 45, 66, 75, 90, 170, 802**
pass 3: **2, 24, 45, 66, 75, 90, 170, 802**
pass 4: **2, 24, 45, 66, 75, 90, 170, 802**
pass 5: **2, 24, 45, 66, 75, 90, 170, 802**
pass 6: **2, 24, 45, 66, 75, 90, 170, 802**
pass 7: **2, 24, 45, 66, 75, 90, 170, 802**



In-Place Quick Sort

- Use the last element in the list as pivot for ascending order sorting.
- In the Partition Step
 - Use 2 cursors (L cursor and R cursor)
 - L cursor scans the list from left to right
 - R cursor scans the list from right to left
 - swap left key $>$ pivot with right key $<$ pivot
 - When the L cursor and R cursor cross, move only one of them.
 - Move the L cursor until it finds a key $>$ pivot or reaches the pivot
 - Swap the last key found with the pivot, if necessary



In-Place Quick Sort: Concepts

sort the list: (pivot=last element)

85, 24, 63, 45, 17, 31, 96, 50

L \longrightarrow

\longleftarrow **R**

look for key > pivot

look for key < pivot

85, 24, 63, 45, 17, 31, 96, 50

swap two keys so that

key < pivot will go to the left and

key > pivot will go to the right



In-Place Quick Sort Example

sort the list:

85, 24, 63, 45, 17, 31, 96, 50

L →

← **R**

pass 1: **85**, 24, 63, 45, 17, **31**, 96, **50**

31, 24, 63, 45, 17, **85**, 96, **50**

exchange 31 and 85

and continue search

L →
← **R**



In-Place Quick Sort Example (cont'd)

sort the list:

85, 24, 63, 45, 17, 31, 96, 50

L →

← R

pass 1: 85, 24, 63, 45, 17, 31, 96, 50

31, 24, 63, 45, 17, 85, 96, 50

31, 24, 63, 45, 17, 85, 96, 50

31, 24, 17, 45, 63, 85, 96, 50

L →

← R

R stops, but L continues until it finds a key > pivot
The last key found by L is 63.

It is > pivot (50), so swap it with the pivot.



Exercise: In-Place Quick Sort

- Continue and complete the example

sort the list: (pivot=last element)

85, 24, 63, 45, 17, 31, 96, 50

pass 1
result **31, 24, 17, 45, 50, 85, 96, 63**

pass 2: **31, 24, 17, 45, 50, 63, 96, 85**

pass 3: **17, 24, 31, 45, 50, 63, 85, 96**

pass 4: **17, 24, 31, 45, 50, 63, 85, 96**



Exercise (show all the steps)

sort the list: (pivot=last element)
(* do not split up the P list already formed *)

85, 17, 63, 45, 17, 31, 85, 50



Exercise (show all the steps)

sort the list: (pivot=last element)
(* do not split up the P list already formed *)

85, 17, 63, 45, 17, 31, 85, 50

Pass 1: 31, 17, 17, 45, 50, 85, 85, 63

Pass 2: 31, 17, 17, 45, 50, 63, 85, 85

Pass 3: 17, 17, 31, 45, 50, 63, 85, 85

Pass 4: 17, 17, 31, 45, 50, 63, 85, 85



Exercise (show all the steps)

- The first element of the list may be selected as pivot to do descending order sorting.
- In this case, the L cursor and R cursor is reversed in direction.
- ** This is left as exercise.

sort the list: (pivot=first element)

85, 24, 63, 45, 17, 31, 96, 50



Quick Sort: Pivot Selection

- First or Last
- Median of Three
 - Select 3 keys, and select the middle (sized) key.
 - first 3, last 3, random 3
 - (first, last, middle)
 - median of median of three



Quick Sort: Performance and Problems

- Partition $O(n)$
 - n comparisons & copying of keys on each pass
- $O(n \log_2 n)$ (Best, Average)
 - about $\log_2 n$ passes
 - if the distribution of the keys is reasonably balanced around the pivot
- Quicksort: (**Not** in-place sort. Use in-place quicksort)
 - $O(n)$ extra storage space for the temporary lists
- Terrible on an already sorted or nearly sorted list
 - worst case: $O(n^2)$
- Randomized quicksort
 - Use quick sort with randomized pivot selection to avoid the worst case performance
- Unstable sort
 - Partition may swap the relative orders according to pivot's position for duplicate elements

Quick Sort: Example code

```
#include <stdio.h>

// function to swap values of two variables
void swap(int *a, int *b) {
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}

int partition(int a[], int start, int end) {
    int pivot = a[end];
    int i = start-1;
    int j;

    for(j=start; j<=end-1; j++) {
        if(a[j]<=pivot) {
            i=i+1;
            swap(&a[i], &a[j]);
        }
    }
    swap(&a[i+1], &a[end]);
    return i+1;
}
```

```
void quicksort(int a[], int start, int end) {
    if(start < end) {
        int q = partition(a, start, end);
        quicksort(a, start, q-1);
        quicksort(a, q+1, end);
    }
}

int main() {
    int a[] = {4, 8, 1, 3, 10, 9, 2, 11, 5, 6};
    quicksort(a, 0, 9);

    //printing array
    int i;
    for(i=0; i<10; i++) {
        printf("%d ", a[i]);
    }
    printf("\n");
    return 0;
}
```



Quick Sort vs. Merge Sort

	quick sort	merge sort
worst case	$O(n^2)$	$O(n \log_2 n)$
best case	$O(n \log_2 n)$	$O(n \log_2 n)$
average	$O(n \log_2 n)$	$O(n \log_2 n)$
storage	$O(n)$	$O(n)$
stable?	no	yes



End of Lecture
