

**CSE1303 Part A**  
**Data Structures and Algorithms**  
**Summer Semester 2003**

**Lecture A16 – Advanced Sorting**

Kymerly Fergusson

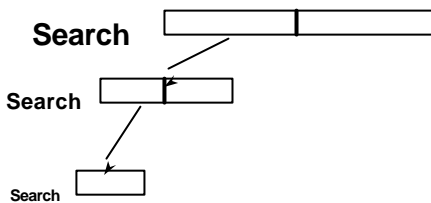
**Overview**

- Divide and Conquer
- Merge Sort
- Quick Sort

2

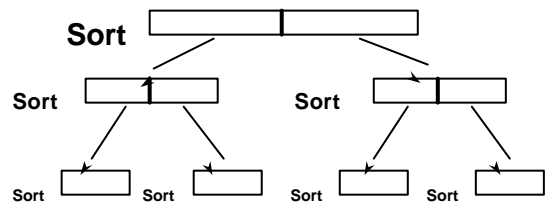
**Divide and Conquer**

**Recall: Binary Search**



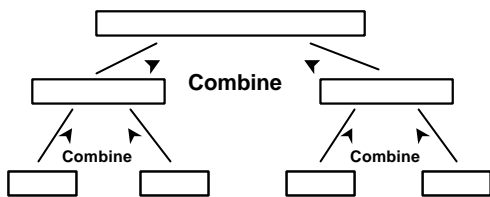
3

**Divide and Conquer**



4

**Divide and Conquer**

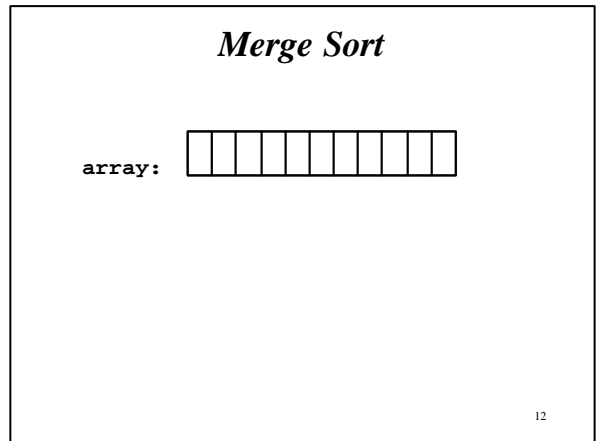
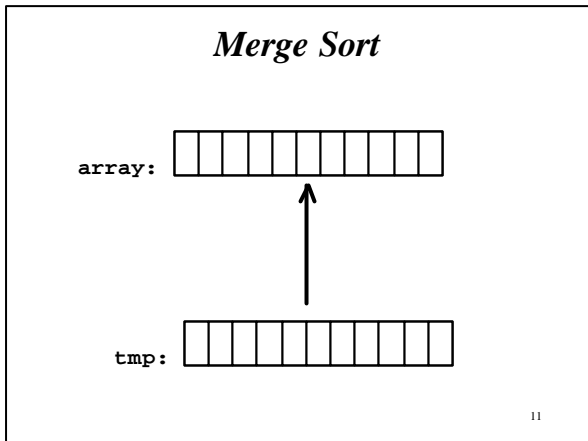
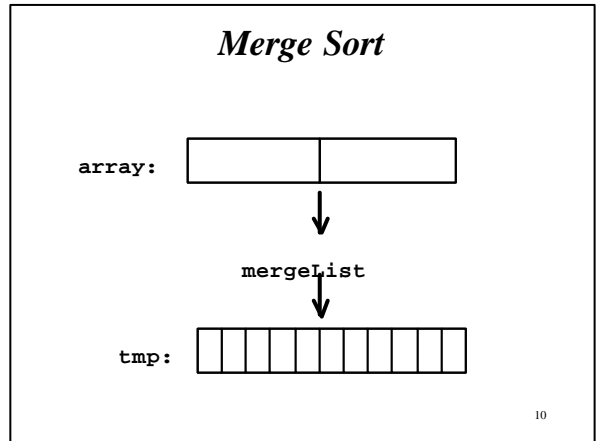
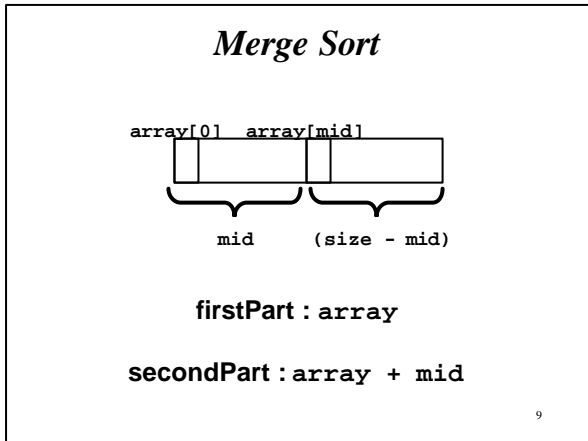
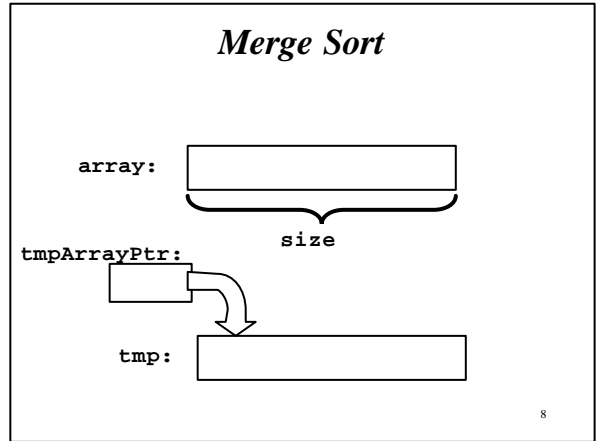
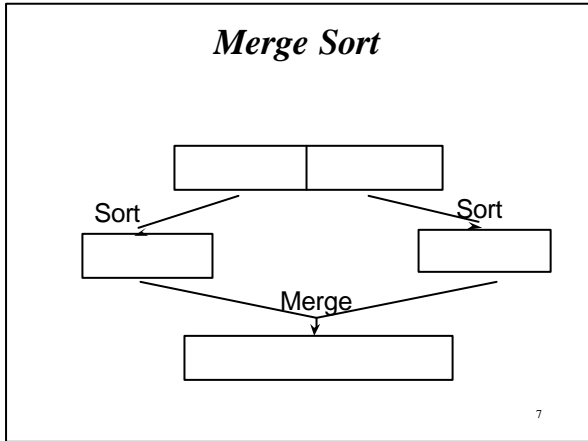


5

**Divide and Conquer**

```
module sort(array)
{
  if (size of array > 1)
  {
    split(array, firstPart, secondPart)
    sort(firstPart)
    sort(secondPart)
    combine(firstPart, secondPart)
  }
}
```

6



```
void mergeSort(float array[], int size)
{
    int* tmpArrayPtr = (int*)malloc(size*sizeof(int));

    if (tmpArrayPtr != NULL)
    {
        mergeSortRec(array, size, tmpArrayPtr);
    }
    else
    {
        fprintf(stderr, "Not enough memory to sort list.\n");
        exit(1);
    }

    free(tmpArrayPtr);
}
```

13

```
void mergeSortRec(float array[], int size, float tmp[])
{
    int i;
    int mid = size/2;

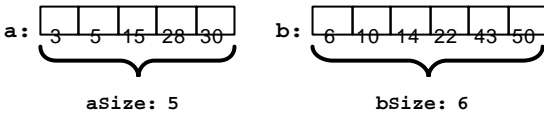
    if (size > 1)
    {
        mergeSortRec(array, mid, tmp);
        mergeSortRec(array+mid, size-mid, tmp);

        mergeArrays(array, mid, array+mid, size-mid, tmp);

        for (i = 0; i < size; i++)
        {
            array[i] = tmp[i];
        }
    }
}
```

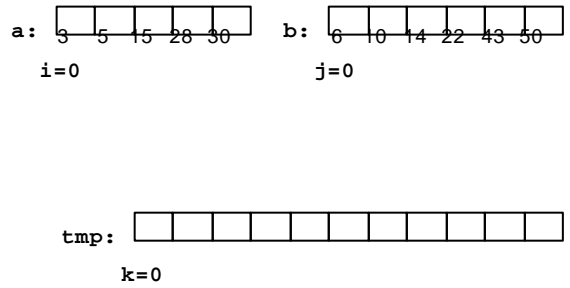
14

**Example: mergeArrays**



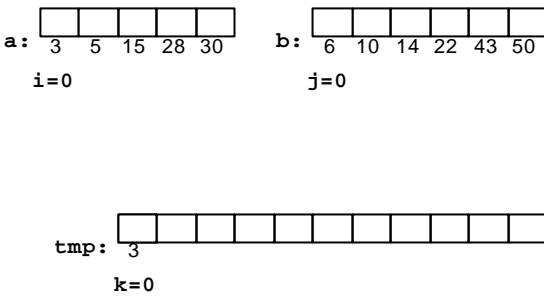
15

**Example: mergeArrays**



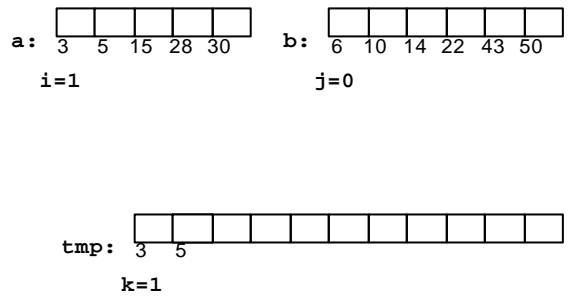
16

**Example: mergeArrays**



17

**Example: mergeArrays**



18

**Example: mergeArrays**

a: [ 3 | 5 | 15 | 28 | 30 ]    b: [ 6 | 10 | 14 | 22 | 43 | 50 ]  
 i=2                                  j=0

tmp: [ 3 | 5 | 6 |   |   |   |   |   |   |   ]  
 k=2    etc...

19

**Example: mergeArrays**

a: [ 3 | 5 | 15 | 28 | 30 ]    b: [ 6 | 10 | 14 | 22 | 43 | 50 ]  
 i=5                                  j=4

Done.

tmp: [ 3 | 5 | 6 | 10 | 14 | 15 | 22 | 28 | 30 | 43 | 50 ]  
 k=9

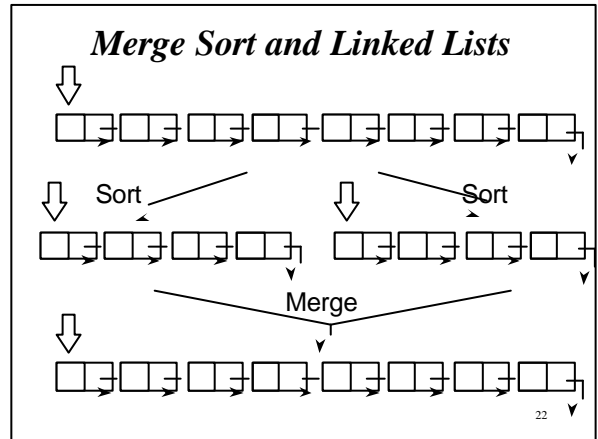
20

```

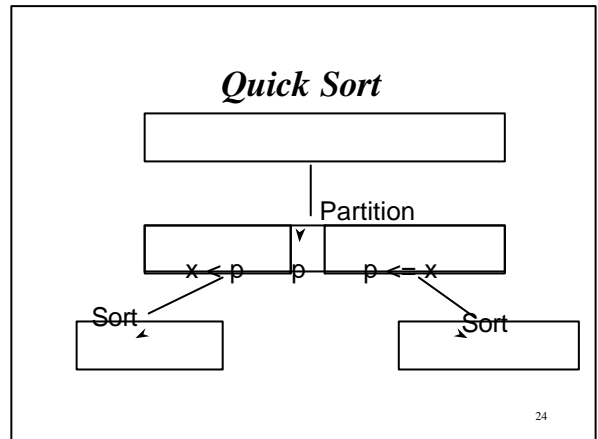
void
mergeArrays(float a[],int aSize,float b[],int bSize,float tmp[])
{
    int k, i = 0, j = 0;

    for (k = 0; k < aSize + bSize; k++)
    {
        if (i == aSize) {
            tmp[k] = b[j];
            j++;
        }
        else if (j == bSize) {
            tmp[k] = a[i];
            i++;
        }
        else if (a[i] <= b[j]) {
            tmp[k] = a[i];
            i++;
        }
        else {
            tmp[k] = b[j];
            j++;
        }
    }
}
    
```

21



- Merge Sort Analysis**
- Most of the work is in the merging.
  - Takes  $O(n \log(n))$
  - Uses more space than other sorts.
  - This is typically the method that you would naturally use when sorting a pile of books, CDs cards, etc.
- 23



**Example: Quick Sort**

array: 

5	89	35	10	24	15	37	13	20	17	70
---	----	----	----	----	----	----	----	----	----	----

size: 11

25

**Example: Quick Sort**

array: 

5	89	35	14	24	15	37	13	20	7	70
---	----	----	----	----	----	----	----	----	---	----

“pivot element”

26

**Example: Quick Sort**

array: 

5	89	35	14	24	15	37	13	20	7	70
---	----	----	----	----	----	----	----	----	---	----

partition: 

5	89	35	14	24	15	37	13	20	7	70
---	----	----	----	----	----	----	----	----	---	----

7	14	5	13	15	35	37	89	20	24	70
---	----	---	----	----	----	----	----	----	----	----

index: 4

27

**Example: Quick Sort**

index: 4

array: 

7	14	5	13	15	35	37	89	20	24	70
---	----	---	----	----	----	----	----	----	----	----

index                      (size - index - 1)

28

**Example: Quick Sort**

array[0]                      array[index + 1]

7	14	5	13	15	35	37	89	20	24	70
---	----	---	----	----	----	----	----	----	----	----

index                      (size - index - 1)

**firstPart : array**

**secondPart : array + index + 1**

29

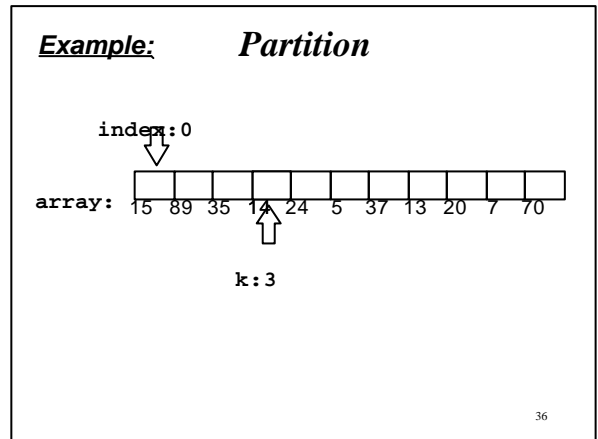
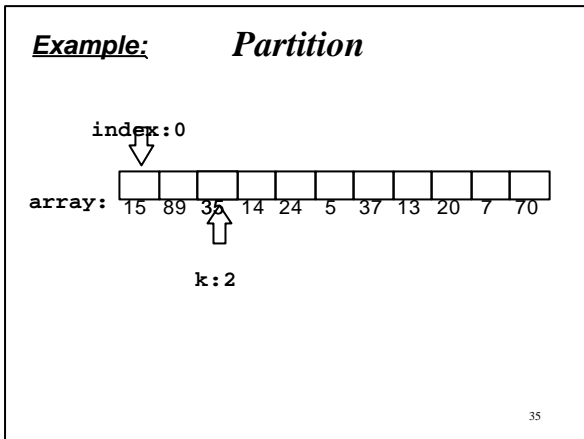
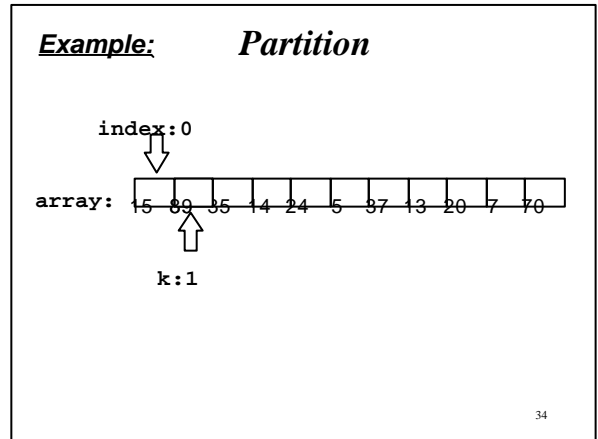
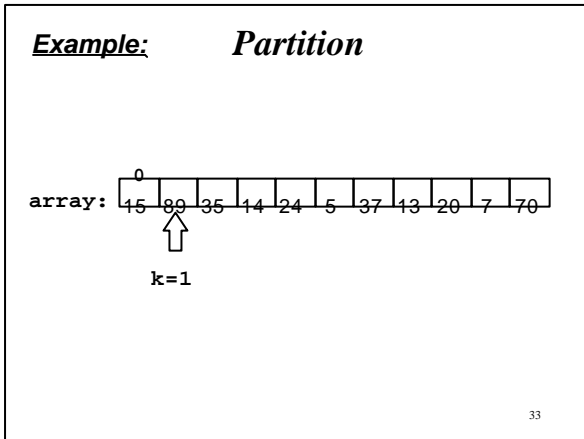
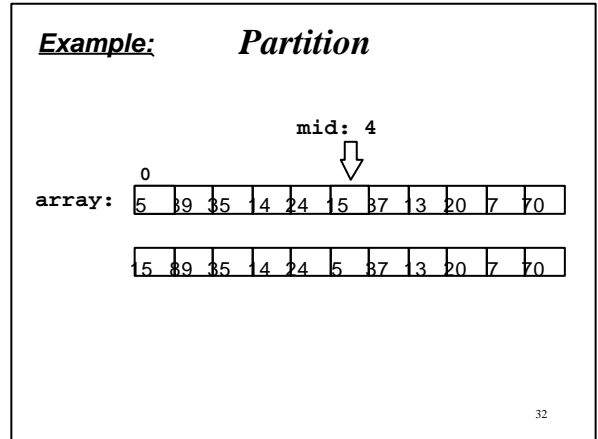
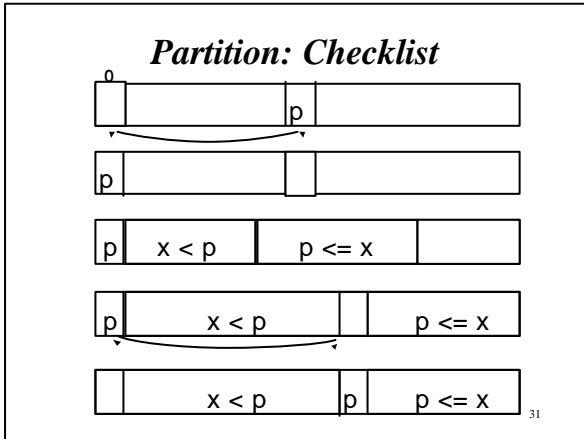
**Quick Sort**

```

void quickSort(float array[], int size)
{
    int index;

    if (size > 1)
    {
        index = partition(array, size);
        quickSort(array, index);
        quickSort(array+index+1, size - index - 1);
    }
}
    
```

30



**Example: Partition**

index:1

array: [15 | 14 | 35 | 89 | 24 | 5 | 37 | 13 | 20 | 7 | 70]

k:3

37

**Example: Partition**

index:1

array: [15 | 14 | 35 | 89 | 24 | 5 | 37 | 13 | 20 | 7 | 70]

k:4

38

**Example: Partition**

index:1

array: [15 | 14 | 35 | 89 | 24 | 5 | 37 | 13 | 20 | 7 | 70]

k:5

39

**Example: Partition**

index:2

array: [15 | 14 | 35 | 89 | 24 | 35 | 37 | 13 | 20 | 7 | 70]

k:5 etc...

40

**Example: Partition**

index:4

array: [15 | 14 | 5 | 13 | 7 | 35 | 37 | 89 | 20 | 24 | 70]

k:11

41

**Example: Partition**

index:4

array: [15 | 14 | 5 | 13 | 7 | 35 | 37 | 89 | 20 | 24 | 70]

42

**Example: Partition**

index: 4

array: [7 | 14 | 5 | 13 | 15 | 35 | 37 | 89 | 20 | 24 | 70]

$x < 15$                        $15 \leq x$

43

**Example:**

pivot now in correct position

array: [7 | 4 | 5 | 3 | 15 | 35 | 37 | 89 | 20 | 24 | 70]

$x < 15$                        $15 \leq x$

44

**Example:**

Sort                      Sort

[7 | 14 | 5 | 13]                      [35 | 37 | 89 | 20 | 24 | 70]

45

```
int partition(float array[], int size)
{
    int k;
    int mid = size/2;
    int index = 0;

    swap(array, array+mid);

    for (k = 1; k < size; k++)
    {
        if (list[k] < list[0])
        {
            index++;
            swap(array+k, array+index);
        }
    }

    swap(array, array+index);

    return index;
}
```

46

**Quick Sort Analysis**

- Most of the work done in partitioning.
- Need to be careful of choice of pivot.
  - Example: 2, 4, 6, 7, 3, 1, 5
- Average case takes  $O(n \log(n))$  time.
- Worst case takes  $O(n^2)$  time.

47

**Revision**

- Merge Sort
- Quick Sort

**Revision: Reading**

- Kruse 7.6 – 7.8
- Standish 13.4
- Langsam 6.2, 6.4
- Sedgewick 9, 12

**Preparation**

Next lecture: Revision

- Revise lecture notes for Part A
- Come with questions!

48