

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

**Lecture A16 – Advanced Sorting**

Kymerly Fergusson

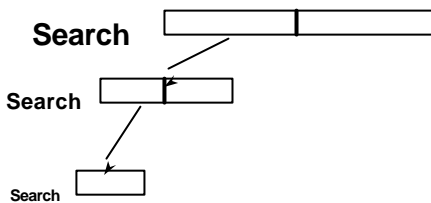
**Overview**

- Divide and Conquer
- Merge Sort
- Quick Sort

2

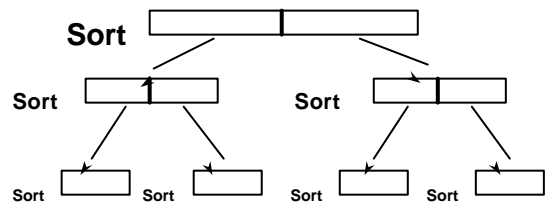
**Divide and Conquer**

**Recall: Binary Search**



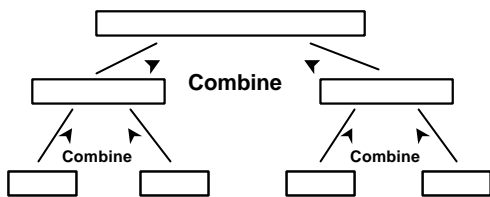
3

**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

### *Divide and Conquer*

```

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

7

### *Divide and Conquer*

```

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

8

### *Divide and Conquer*

```

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

9

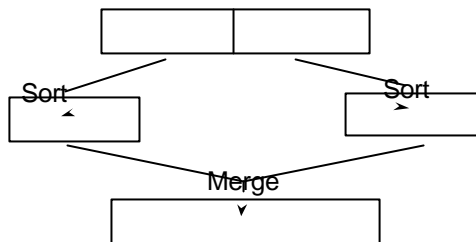
### *Divide and Conquer*

```

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

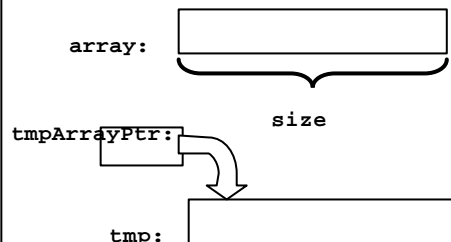
10

### *Merge Sort*



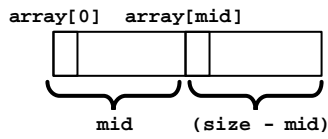
11

### *Merge Sort*



12

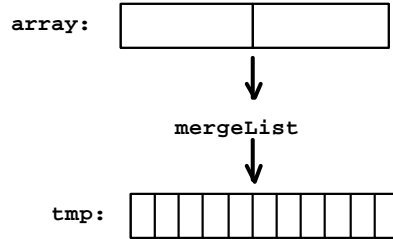
### Merge Sort



firstPart : array  
secondPart : array + mid

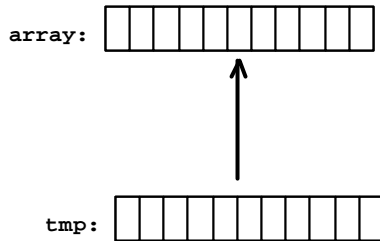
13

### Merge Sort



14

### Merge Sort



15

### Merge Sort



16

```
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);
}
```

17

```
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];
        }
    }
}
```

18

**Example: mergeArrays**

a: 

3	5	15	28	30
---	---	----	----	----

    b: 

6	10	14	22	43	50
---	----	----	----	----	----

        └──────────┘                  └──────────┘

        aSize: 5                      bSize: 6

tmp: 

--	--	--	--	--	--	--	--	--	--

19

**Example: mergeArrays**

a: 

3	5	15	28	30
---	---	----	----	----

    b: 

6	10	14	22	43	50
---	----	----	----	----	----

        i=0                                  j=0

tmp: 

--	--	--	--	--	--	--	--	--	--

        k=0

20

**Example: mergeArrays**

a: 

3	5	15	28	30
---	---	----	----	----

    b: 

6	10	14	22	43	50
---	----	----	----	----	----

        i=0                                  j=0

tmp: 

3									
---	--	--	--	--	--	--	--	--	--

        k=0

21

**Example: mergeArrays**

a: 

3	5	15	28	30
---	---	----	----	----

    b: 

6	10	14	22	43	50
---	----	----	----	----	----

        i=1                                  j=0

tmp: 

3	5								
---	---	--	--	--	--	--	--	--	--

        k=1

22

**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

23

**Example: mergeArrays**

a: 

3	5	15	28	30
---	---	----	----	----

    b: 

6	10	14	22	43	50
---	----	----	----	----	----

        i=2                                  j=1

tmp: 

3	5	6	10						
---	---	---	----	--	--	--	--	--	--

        k=3

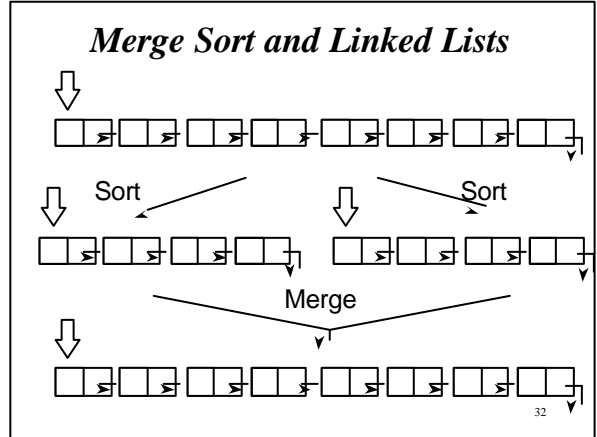
24



```

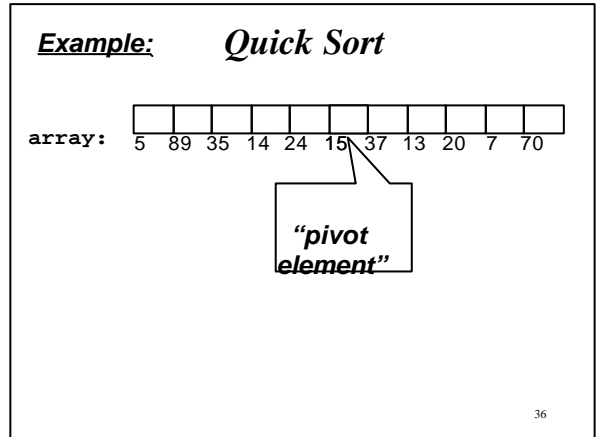
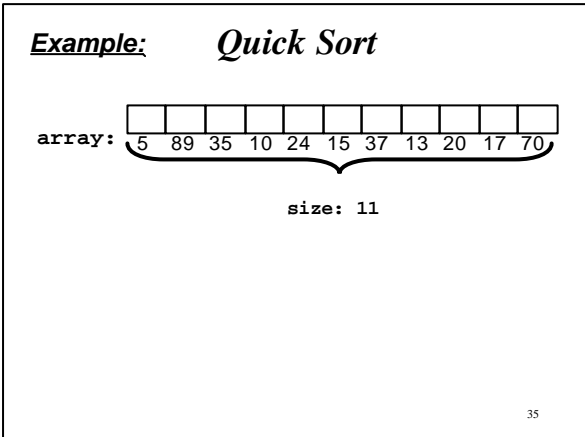
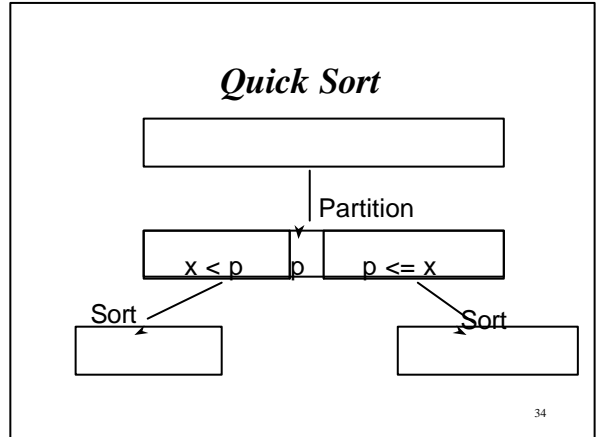
void
mergeArrays(float a[],int aSize,float b[],int bSize,float tmp[])
{
    int k, i = 0, j = 0;

    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++;
        }
    }
}
    
```



### Merge Sort Analysis

- Most of the work is in the merging.
- Takes  $O(n \log(n))$
- Uses more space than other sorts.
- Useful for linked lists.



**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

37

**Example: Quick Sort**

index: 4

array: [7 | 14 | 5 | 13 | 15 | 35 | 87 | 89 | 20 | 24 | 70]

index (size - index - 1)

38

**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

39

**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);
    }
}
```

40

**Partition: Checklist**

41

**Example: Partition**

mid: 4

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

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

42

**Example: Partition**

array: 

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

index: 0 (points to 15)

k: 1 (points to 89)

43

**Example: Partition**

array: 

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

index: 0 (points to 15)

k: 1 (points to 89)

44

**Example: Partition**

array: 

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

index: 0 (points to 15)

k: 2 (points to 35)

45

**Example: Partition**

array: 

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

index: 0 (points to 15)

k: 3 (points to 14)

46

**Example: Partition**

array: 

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

index: 1 (points to 89)

k: 3 (points to 14)

47

**Example: Partition**

array: 

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

index: 1 (points to 14)

k: 4 (points to 24)

48

**Example: Partition**

index: 1

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

k: 5

49

**Example: Partition**

index: 2

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

k: 5

50

**Example: Partition**

index: 2

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

k: 6

51

**Example: Partition**

index: 2

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

k: 7 etc...

52

**Example: Partition**

index: 4

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

k: 11

53

**Example: Partition**

index: 4

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

54

**Example: Partition**

index: 4

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

$x < 15$                        $15 \leq x$

55

**Example:**

pivot now in correct position

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

$x < 15$                        $15 \leq x$

56

**Example:**

Sort                      Sort

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

57

```
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;
}
```

58

**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.

59

**Revision**

- Merge Sort
- Quick Sort

60

***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!

61