# Divide & Conquer
# Sort

1

## Divide and Conquer

```
void Sort(List *list)
{
    if (list has length >= 1)
    {
        Partition list into lowlist, highlist;
        Sort(lowlist);
        Sort(highlist);
        Combine(lowlist, highlist);
    }
}
```

DESIGN & ANALYSIS OF ALGORITHM


ALG00S, javed@kent.edu

# Merge Sort Example

DESIGN & ANALYSIS OF ALGORITHM

**Mergesort:**

We chop the list into two sublists of sizes as nearly equal as possible and then sort them separately. Afterward, we carefully merge the two sorted sublists into a single sorted list.

```
void Sort(List *list)
{
   if (list has length >= 1)
   {
      Partition list into lowlist, highlist;
      Sort(lowlist);
      Sort(highlist);
      Combine(lowlist, highlist);
   }
}
```

- Let's Sort:

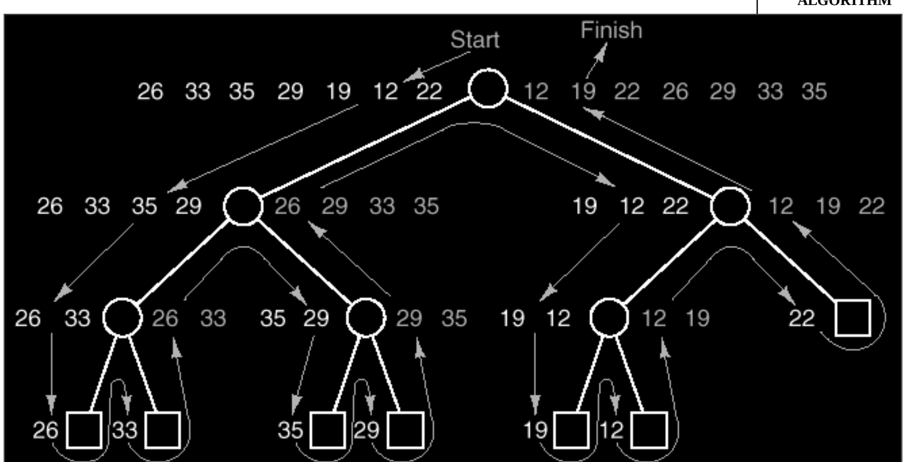
26 33 35 29 19 12 22

**Note: When we cannot divide into two equal list we will make the first one large.**



# Recursion Tree of Merge Sort

DESIGN & ANALYSIS OF ALGORITHM



# Quick Sort Example

DESIGN &
ANALYSIS OF
ALGORITHM

**Quicksort:**

We first choose some key from the list for which, we hope, about half the keys will come before and half after. Call this key the ***pivot***. Then we partition the items so that all those with keys less than the pivot come in one sublist, and all those with greater keys come in another. Then we sort the two reduced lists separately, put the sublists together, and the whole list will be in order.

```
void Sort(List *list)
{
    if (list has length >= 1)
    {
        Partition list into lowlist, highlist;
        Sort(lowlist);
        Sort(highlist);
        Combine(lowlist, highlist);
    }
}
```

- Let's Sort:

26 33 35 29 19 12 22

**Note: Let us pick the first element on the list as the pivot.**



# Execution Trace of Quick Sort

DESIGN &
ANALYSIS OF
ALGORITHM

Sort (26, 33, 35, 29, 12, 22)

Partition into (19, 12, 22) and 33, 35, 29); pivot = 26
Sort (19, 12, 22)

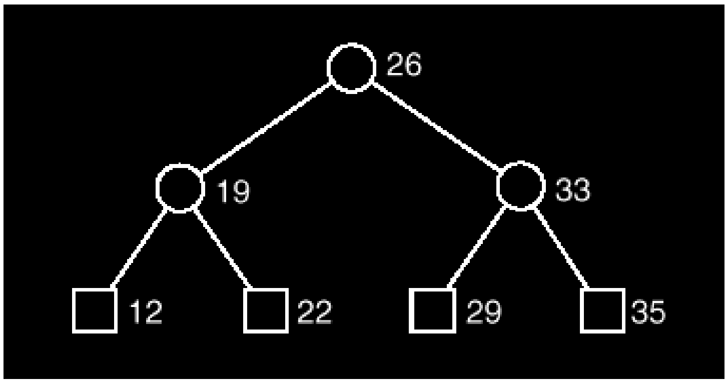
Partition into (12) and (22); pivot = 19
Sort (12)
Sort (22)
Combine into (12, 19, 22)

Sort (33, 35, 29)

Partition into (29) and (35); pivot = 33
Sort (29)
Sort (35)
Combine into (29, 33, 35)

Combine into (12, 19, 22, 26, 29, 33 35)

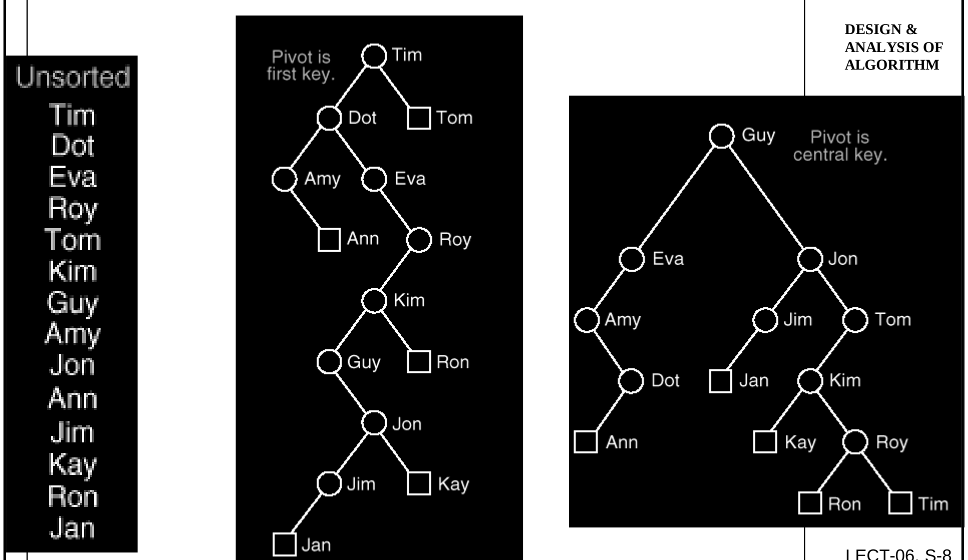
# Recursion Tree of Quick Sort




DESIGN & ANALYSIS OF ALGORITHM



# Another Example with QS




DESIGN & ANALYSIS OF ALGORITHM

# Main for Merge Sort

DESIGN & ANALYSIS OF ALGORITHM

```
void MergeSort(List *list)
{
    List secondhalf;    /* holds the second half of the list after
division */

    if (ListSize(list) > 1) {       /* Is there a need to sort? */
        Divide(list, &secondhalf);  /* Divide the list in half. */
        MergeSort(list);            /* Sort the first half.     */
        MergeSort(&secondhalf);     /* Sort the second half.    */
        Merge(list, &secondhalf, list); /* Merge the two sorted
sublists. */
    }
}
```

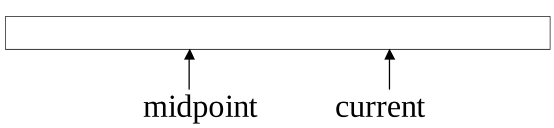


# Divide (Linked List)

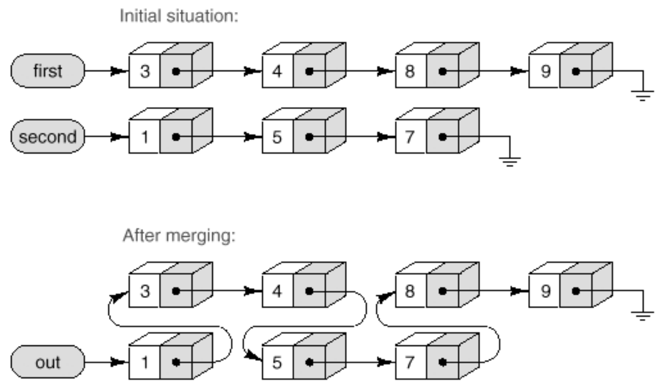
DESIGN & ANALYSIS OF ALGORITHM

```
void Divide(List *list, List *secondhalf)
{
    ListNode *current, *midpoint;

    if ((midpoint = list->head) == NULL)
        secondhalf->head = NULL;
    else {
        for (current = midpoint->next; current; ) {
            current = current->next;
            if (current) {
                midpoint = midpoint->next;
                current = current->next;
            }
        }
        secondhalf->head = midpoint->next;
        midpoint->next = NULL;
    }
}
```



# Merging Two Sorted List




DESIGN & ANALYSIS OF ALGORITHM



# Code for Merge (Linked List)

```
void Merge(List *first, List *second, List *out)
{
    ListNode *p1, *p2;  /* pointers to traverse first and second lists */
    ListNode *lastsorted;   /* always points to last node of sorted list */

    if (!first->head)
        *out = *second;
    else if (!second->head)
        *out = *first;
    else {
        p1 = first->head;   /* First find the head of the merged list. */
        p2 = second->head;
        if (LE(p1->entry.key, p2->entry.key)) {
            *out = *first;
            p1 = p1->next;
        } else {
            *out = *second;
            p2 = p2->next;
        }
        lastsorted = out->head; /* lastsorted gives last entry of merged list. */
        while (p1 && p2)
            if (LE(p1->entry.key, p2->entry.key)) {
                lastsorted->next = p1;
                lastsorted = p1;
                p1 = p1->next;
            } else {
                lastsorted->next = p2;
                lastsorted = p2;
                p2 = p2->next;
            }
        if (p1)                 /* Attach the remaining list. */
            lastsorted->next = p1;
        else
            lastsorted->next = p2;
    }
}
```

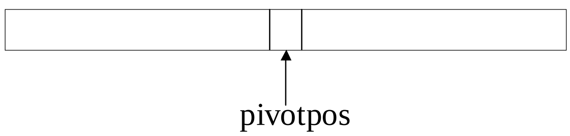
DESIGN & ANALYSIS OF ALGORITHM

# Quick Sort for Contiguous List

DESIGN & ANALYSIS OF ALGORITHM

```
void RecQuickSort(List *list, Position low, Position
high)
{
    Position pivotpos;      /* position of the pivot
after partitioning */

    if (low < high) {
        pivotpos = Partition(list, low, high);
        RecQuickSort(list, low, pivotpos - 1);
        RecQuickSort(list, pivotpos + 1, high);
    }
}
```



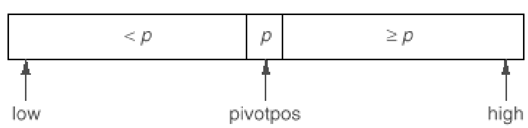



ALG00S, javed@kent.edu
Javed I. Khan@1999

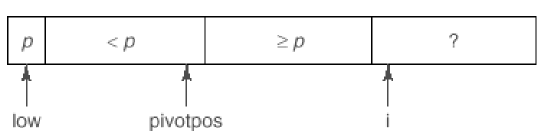

# Partitioning in Quick Sort


DESIGN & ANALYSIS OF ALGORITHM


**Goal (postcondition):**




**Loop invariant:**

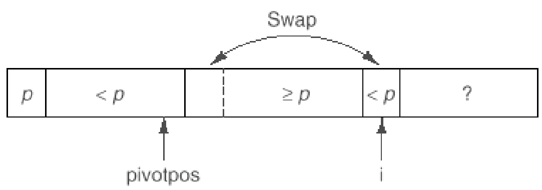




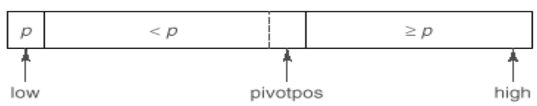

ALG00S, javed@kent.edu
Javed I. Khan@1999

## Partitioning in Quick Sort

**Restore the invariant:**




**Final position:**




DESIGN & ANALYSIS OF ALGORITHM

LECT-06, S-15



## Partition (code)

DESIGN & ANALYSIS OF ALGORITHM

```
Position Partition(List *list, Position low, Position high)
{
    ListEntry pivot;
    Position i, lastsmall, pivotpos;

    Swap(low, (low + high) / 2, list);
    pivot= list->entry[low];
    pivotpos = low;
    for (i = low + 1; i <= high; i++)
        if (LT(list->entry[i].key, pivot.key)) {
            Swap(++pivotpos, i, list);
         lastsmall++;
       }/* Move large entry to right and small to left. */
    Swap(low, pivotpos, list);
    return pivotpos;
}
```

LECT-06, S-16

# Analysis of Quick & Merge Sort

17

DESIGN & ANALYSIS OF ALGORITHM

## Need Volunteer!

- To keep various performances of various algorithms.
- Insertion Sort: Worst Case assignments?
- Selection Sort: Worst case comparisons?

*Need Volunteer!*

| | *Selection* | *Insertion (average)* |
|---|---|---|
| *Assignments of entries* | $3.0n + O(1)$ | $0.25n^2 + O(n)$ |
| *Comparisons of keys* | $0.5n^2 + O(n)$ | $0.25n^2 + O(n)$ |

**Worst case of Selection Sort is twice as bad than average case.**

LECT-06, S-18
ALG00S, javed@kent.edu
Javed I. Khan@1999

## Few Results!



$$S_n = 1 + 2 + 3 + ..... + n = \frac{n(n+1)}{2}$$

$$1^2 + 2^2 + 3^2 + .... + n^2 = \frac{n(n+1)(2n+1)}{6}$$

$$1 + a^1 + +a^2 .... + a^m = \frac{a^{m+1} - 1}{(a-1)}$$

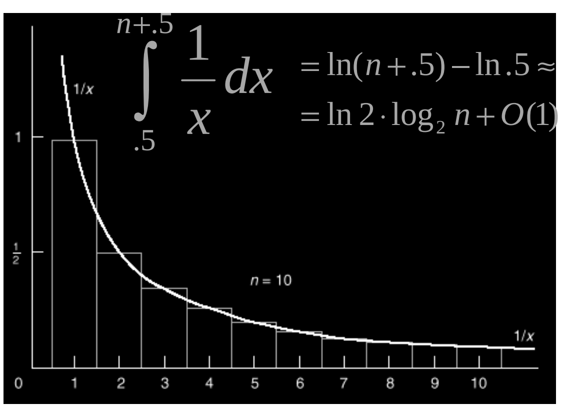
$$\log_b x = \log_a x \cdot \log_b a$$



## Harmonic Numbers



$$H_n = \frac{1}{n} + \frac{1}{n-1} + ... + \frac{1}{3} + \frac{1}{2} + 1$$




$$\int_{.5}^{n+.5} \frac{1}{x} dx \quad = \ln(n+.5) - \ln .5 \approx \ln n + .7$$
$$= \ln 2 \cdot \log_2 n + O(1) = .69 \log_2 n + O(1)$$

*Need Volunteer!*

# Analysis of Merge Sort

DESIGN & ANALYSIS OF ALGORITHM

- Merging two lists of size k requires at most (k-1) comparisons. There are:

$$(n-1)+2\cdot(\frac{n}{2}-1)+4\cdot(\frac{n}{4}-1)+...+n\cdot(\frac{n}{n}-1)$$
$$=n+n+....+n-(2^0+2^1+2^2+2^3+....+2^{\log n-1})$$
$$=n\log n-1\frac{(2^{\log n}-1)}{(2-1)}$$
$$=n\log n-(n-1)=n\log n-n+1$$

- Note this is worst case at exact count!
- Exercise E2 outlines a method which shows average case is:

$$=n\log n-1.1583n+1$$

*Need Volunteers again!*

LECT-06, S-21
ALG00S, javed@kent.edu
Javed I. Khan@1999

# Merge Sort: the ultimate sorting method?

DESIGN & ANALYSIS OF ALGORITHM

- Average Performance:

$$=n\log n-1.1583n+1$$

**QUIZ: Can we improve finding the middle?**

- The lowest bound of any comp. sorting algorithm we have derived:

$$\boxed{\log n!\approx n.\log n-1.44n+O(\log n)}$$

- It is indeed, for linked list in random initial order, it is difficult to surpass.

LECT-06, S-22
ALG00S, javed@kent.edu
Javed I. Khan@1999

# Worst Case Analysis of Quick Sort

DESIGN & ANALYSIS OF ALGORITHM

- Count of Comparisons and Swaps

$c(n) = n - 1 + c(r) + c(n - r - 1)$

- Comparison and Swap counts will be different.
- Comparison Count: Worst Case:

$= 0.5n^2 - 0.5n$

$$c(1) = 0$$
$$c(2) = 1 + c(1) = 1$$
$$c(3) = 2 + c(2) = 2 + 1$$
$$c(4) = 3 + c(3) = 3 + 2 + 1$$
$$c(n) = n - 1 + c(n - 1) = ?$$
$$= \frac{n(n-1)}{2} = .5n^2 - .5n$$

*Need another WC Volunteer again!*

LECT-06, S-23
ALG00S, javed@kent.edu
Javed I. Khan@1999

# WC Analysis of Quick Sort (cont..)

DESIGN & ANALYSIS OF ALGORITHM

- Swap Count: Worst Case:
  - partition function performs one swap inside the loop when the key is smaller than the pivot.
  - It performs two swaps outside the loop
  - In worst case it will perform (n-1)+2=n+1 swaps.

$= 1.5n^2 + 4.5n - 3$

$$S(n) = n + 1 + S(n - 1)$$

- The partition function is called only when n>1, and S(2)=3

$$S(n) = (n+1) + n + (n-1) + .... + 3$$
$$= ????$$
$$= .5n^2 + 1.5n - 1$$

- Number of assignments are three times the number of swaps!

*Need another WC Volunteer again!*

LECT-06, S-24
ALG00S, javed@kent.edu
Javed I. Khan@1999

# Average case Analysis of Quick Sort

- Counting Swaps:
  - The pivot selection will partition the list into two parts. The partition can be anywhere between p=1 to n in the list. For n>1:

$$S(n,p) = (p+1) + S_{avg}(p-1) + S_{avg}(n-p)$$

  - To determine the average case we will allow all possibilities of *p=1 to n* and take an average over the sum of all:

$$S_{avg}(n) = \frac{1}{n}\sum_{p=1}^{n} S(n,p)$$

$$S_{avg}(n) = \frac{n}{2} + \frac{3}{2} + \frac{2}{n}[S(0) + S(1) + \ldots S(n-1)]$$

An equation of this form is called a ***recurrence relation*** because it expresses the answer to a problem in terms of earlier, smaller cases of the same problem.

DESIGN & ANALYSIS OF ALGORITHM



# Solving Recurrence

$$S_a(n) = \frac{n}{2} + \frac{3}{2} + \frac{2}{n}[S_a(0) + S_a(1) + \ldots S_a(n-1)]$$

- From the recurrence we can write:

$$S_a(n-1) = \frac{n-1}{2} + \frac{3}{2} + \frac{2}{n-1}[S_a(0) + S_a(1) + \ldots + S_a(n-2)]$$

- with multiplying n and n-1 respectively and subtracting:

$$n.S_a(n) - (n-1).S_a(n-1) = n + 1 + 2S_a(n-1)$$

$$\frac{S_a(n)}{n+1} = \frac{1}{n} + \frac{S(n-1)}{n}$$

$$\frac{S_a(n)}{n+1} = \frac{1}{n} + \frac{1}{n-1} + \ldots + \frac{1}{3} + \frac{S(2)}{3} = ???$$

DESIGN & ANALYSIS OF ALGORITHM

## Solving Recurrence (contd..)

$$S_a(n) = \frac{n}{2} + \frac{3}{2} + \frac{2}{n}[S_a(0) + S_a(1) + \ldots S_a(n-1)]$$

- From the recurrence we can write:

$$S_a(n-1) = \frac{n-1}{2} + \frac{3}{2} + \frac{2}{n-1}[S_a(0) + S_a(1) + \ldots + S_a(n-2)]$$

- with multiplying n and n-1 respectively and subtracting:

$$n.S_a(n) - (n-1).S_a(n-1) = n + 1 + 2S_a(n-1)$$

$$\frac{S_a(n)}{n+1} = \frac{1}{n} + \frac{S(n-1)}{n}$$

$$\frac{S_a(n)}{n+1} = \frac{1}{n} + \frac{1}{n-1} + \ldots + \frac{1}{3} + \frac{S(2)}{3} = \ln n + O(1)$$

$$S_a(n) \approx .69(n \log n) + O(n)$$

- Each swap needs at least 3 assignments

$$= 2n \log n + O(n)$$

*Need another AC Volunteer again!*

DESIGN & ANALYSIS OF ALGORITHM



## Average case Analysis of Quick Sort

- Counting Comparisons:
  - The partition of a list will make exactly n-1 comparisons:

$$C(n,p) = (n-1) + C_{avg}(p-1) + C_{avg}(n-p)$$

  - Solution can be derived in the exactly same way!

- I have not decided, whether I will make it a part of midterm or a future quiz, but I will advise you to try it out for every step tonight! And the final step will look this:

$$C_{avg}(n) = 2n \ln n + O(n) \approx 1.39 n \log n + O(n)$$

DESIGN & ANALYSIS OF ALGORITHM

# Average Case Comparisons

*Average Case Volunteers Wakeup!*

- The average case for quick sort on contiguous list is one of the most efficient among the known algorithms.
- It requires just 39% more comparisons than mergesort (or best possible case).
- It requires about 100% more assignments than mergesort (in good architecture only 39% more).
  - Considering a 2n space contiguous implementation of the merging algorithm for merge sort.

**QUIZ: How can we ensure that a sorting problem always appears as an average case to a quick sort?**





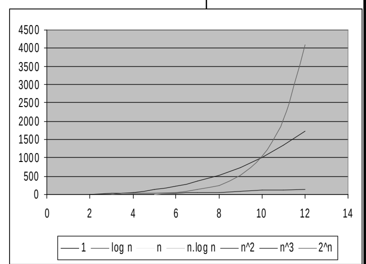
# Suggestion for Midterm

- It will be very important to know the result of the analyses.
- Prepare a table for Worst Case, Average Case, and Best Case for number of comparisons and number of assignments for all the algorithms we are covering.
- You don't have to know all the derivations of the book.
- Learn the general principal behind the proofs.
- but you should go through all the proofs derived in the class.
- One will appear in the Midterm!

# Example Template

DESIGN & ANALYSIS OF ALGORITHM

- A class of students have been given the task of developing a solution for an algorithm to count the number of snow flakes looking through the window. Here are the running time of the solutions.




$$A = 200 \times \log.(\log n) + 32 \times \log.\log.\log(\log n^2)$$

$$B = 2^{\log n} - 3n^{.5} + 10$$

| n | 1 | log n | n | n.log n | n^2 | n^3 | 2^n |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 1 | 1 | 2 |
| 10 | 1 | 3.321928 | 10 | 33.21928 | 100 | 1000 | 1024 |
| 100 | 1 | 6.643856 | 100 | 664.3856 | 10000 | 1000000 | 1.26765E+30 |
| 1000 | 1 | 9.965784 | 1000 | 9965.784 | 1000000 | 1000000000 | 1.0715E+301 |

**Quiz Box**


ALG00S, javed@kent.edu
Javed I. Khan@1999