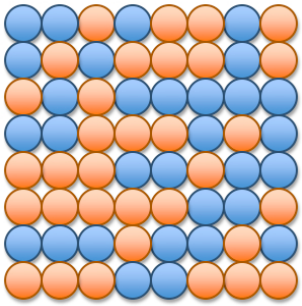


UNIT 5C
Merge Sort

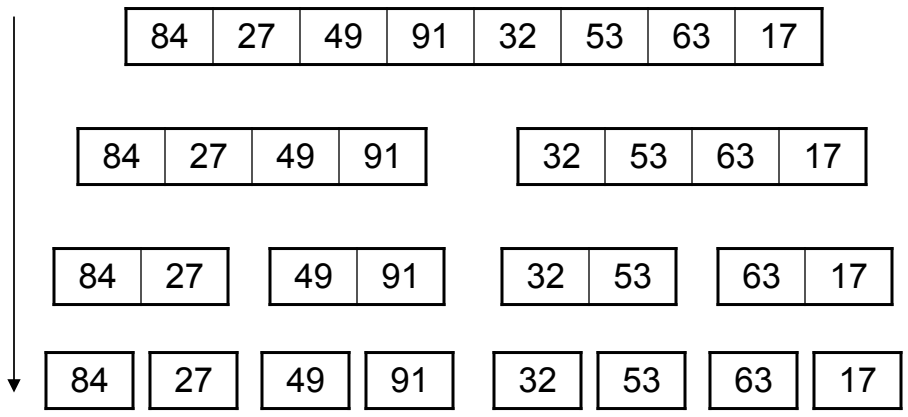
Divide and Conquer

- In the military: strategy to gain or maintain power
- In computation:
  - Divide the problem into "simpler" versions of itself.
  - Conquer each problem using the same process (usually recursively).
  - Combine the results of the "simpler" versions to form your final solution.
- Examples: Towers of Hanoi, fractals, Binary Search, Merge Sort

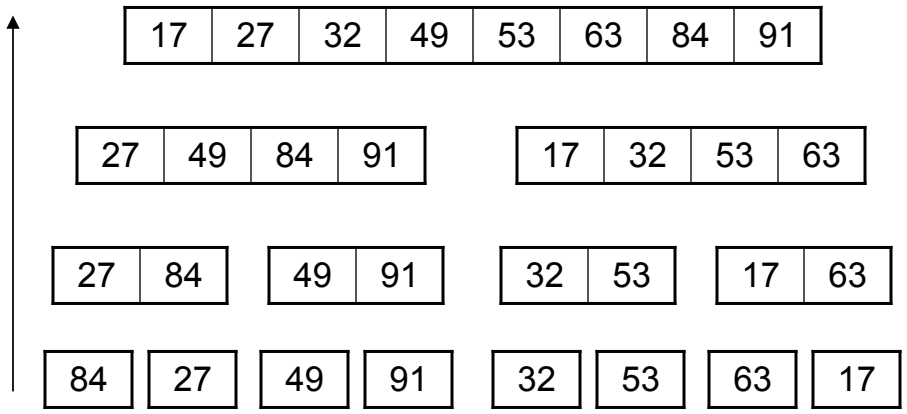
Merge Sort

- Required: List L of n elements.
- Result: Returns a new list containing the same elements in non-decreasing order.
- General algorithm for merge sort:
  1. Sort the first half using merge sort. (recursive!)
  2. Sort the second half using merge sort. (recursive!)
  3. Merge the two sorted halves to obtain the final sorted list.

Divide (Split)



Conquer (Merge)



Example 1: Merge

```
list a          list b          list c
 0  1  2  3     0  1  2  3     0  1  2  3  4  5  6  7
12 44 58 62    29 31 74 80    12

 0  1  2  3     0  1  2  3     0  1  2  3  4  5  6  7
12 44 58 62    29 31 74 80    12 29

 0  1  2  3     0  1  2  3     0  1  2  3  4  5  6  7
12 44 58 62    29 31 74 80    12 29 31

 0  1  2  3     0  1  2  3     0  1  2  3  4  5  6  7
12 44 58 62    29 31 74 80    12 29 31 44
```

Example 1: Merge (cont'd)

list a				list b				list c							
0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
12	44	58	62	29	31	74	80	12	29	31	44	58			
0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
12	44	58	62	29	31	74	80	12	29	31	44	58	62		
0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
12	44	58	62	29	31	74	80	12	29	31	44	58	62	74	80

Example 2: Merge

list a				list b				list c							
0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
58	67	74	90	19	26	31	44	19							
0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
58	67	74	90	19	26	31	44	19	26						
0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
58	67	74	90	19	26	31	44	19	26	31					
0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
58	67	74	90	19	26	31	44	19	26	31	44				
0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
58	67	74	90	19	26	31	44	19	26	31	44	58	67	74	90

Merge

- Required: Two lists a and b.
  - Each list must be sorted already in non-decreasing order.
- Result: Returns a new list containing the same elements merged together into a new list in non-decreasing order.
- We'll need two variables to keep track of where we are in lists a and b: index_a and index_b.

1. Set index_a equal to 0.
2. Set index_b equal to 0.
3. Create an empty list c.

Merge (cont'd)

4. While index_a < the length of list a and index_b < the length of list b, do the following:

   a. If a[index_a] ≤ b[index_b], then do the following:

      i. append a[index_a] on to the end of list c

      ii. add 1 to index_a

   Otherwise, do the following:

      i. append b[index_b] on to the end of list c

      ii. add 1 to index_b

Merge (cont'd)

(Once we finish step 4, we've added all of the elements of either list a or list b to list c. The other list still has some elements left that need to be added to list c.)

5. If index_a < the length of list a, then:

   append all remaining elements of list a on to the end of list c

   Otherwise:

   append all remaining elements of list b on to the end of list c

6. Return list c as the result.

Merge in Python

```
def merge(a, b):
    index_a = 0
    index_b = 0
    c = []
    while index_a < len(a) and index_b < len(b):
        if a[index_a] <= b[index_b]:
            c.append(a[index_a])
            index_a = index_a + 1
        else:
            c.append(b[index_b])
            index_b = index_b + 1
```

Merge in Python (cont'd)

```
if index_a < len(a):
    for i in range(index_a, len(a)):
        c.append(a[i])
else:
    for i in range(index_b, len(b)):
        c.append(b[i])
return c
```


Merge Sort: Base Case

- General algorithm for merge sort:
  1. Sort the first half using merge sort. (recursive!)
  2. Sort the second half using merge sort. (recursive!)
  3. Merge the two sorted halves to obtain the final sorted list.
- What is the base case?
  If the list has only 1 element, it is already sorted
  so just return the list as the result.

Merge Sort: Halfway Point

- General algorithm for merge sort:
  1. Sort the first half using merge sort. (recursive!)
  2. Sort the second half using merge sort. (recursive!)
  3. Merge the two sorted halves to obtain the final sorted list.
- How do we determine the halfway point where we want to split *datalist*?

  Halfway point: `len(datalist) // 2`

  First half: `datalist[0:halfway]`

  Second half: `datalist[halfway:len(datalist)]`

Merge Sort in Python

```
def msort(datalist):
    if len(datalist) <= 1:    # base case
        return datalist
    halfway = len(datalist)//2
    list1 = datalist[0:halfway]
    list2 = datalist[halfway:len(datalist)]
    newlist1 = msort(list1)    # recursive!
    newlist2 = msort(list2)    # recursive!
    newlist = merge(newlist1, newlist2)
    return newlist
```

using merge function from earlier

Analyzing Efficiency

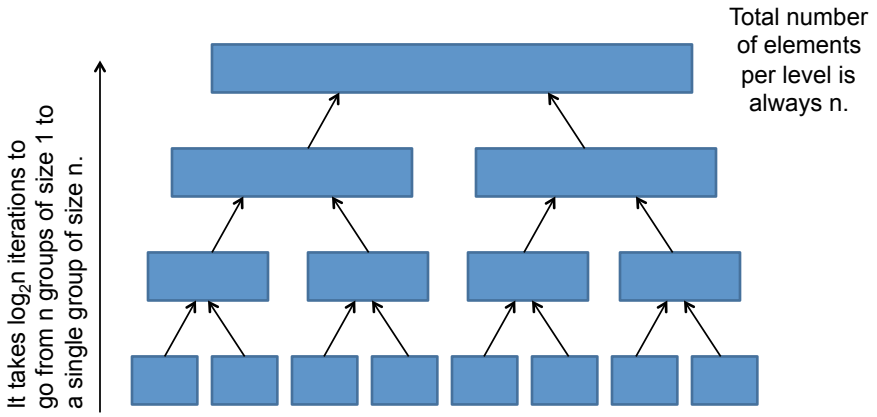
- If you merge two lists of size i/2 into one new list of size i, what is the maximum number of appends that you must do?
  - Clearly, each element must be appended to the new list at some point, so the total number of appends is i.
- Merging n lists of size 1 into n/2 lists of size 2: n
  Merging n/2 lists of size 2 into n/4 lists of size 4: n
  Merging n/4 lists of size 4 into n/8 lists of size 8: n
  ...
  Merging 2 lists of size n/2 into 1 list of size n: n

How many group merges?

- How many group merges does it take to go from n groups of size 1 to 1 group of size n?
- Example: Merge sort on 32 elements.
  - Break down to groups of size 1 (base case).
  - Merge 32 lists of size 1 into 16 lists of size 2.
  - Merge 16 lists of size 2 into 8 lists of size 4.
  - Merge 8 lists of size 4 into 4 lists of size 8.
  - Merge 4 lists of size 8 into 2 lists of size 16.
  - Merge 2 lists of size 16 into 1 list of size 32.

  $5 = \log_2 32$
- In general: $\log_2 n$ group merges must occur.

Putting it all together



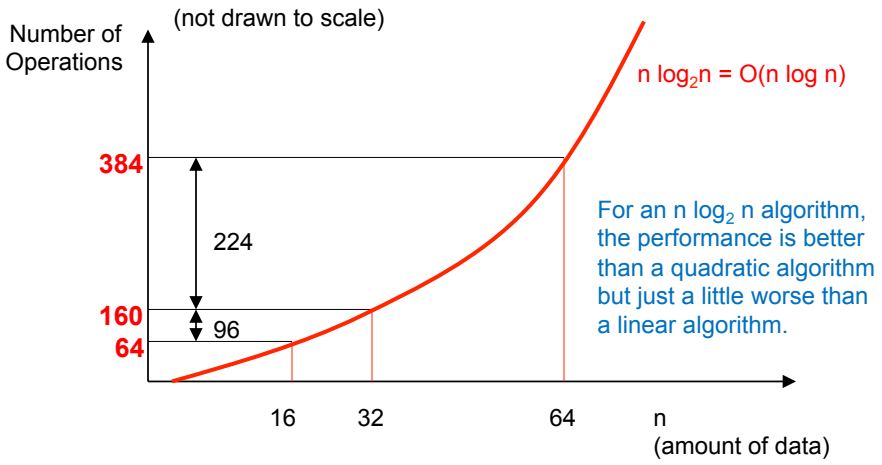
It takes n appends to merge all pairs on one level to the next higher level.

Big O

- In the worst case, merge sort requires O(n log n) time to sort a list with n elements.

Number of operations	Order of Complexity
n $\log_2$n	O(n log n)
4n $\log_{10}$n	O(n log n)
n $\log_2$n + 2n	O(n log n)

O(N log N)



Comparing Insertion Sort to Merge Sort

(Worst Case)

n	isort (n(n+1)/2)	msort (n $\log_2$n)
8	36	24
16	136	64
32	528	160
2^{10}	524,800	10,240
2^{20}	549,756,338,176	20,971,520

For list sizes less than 100, there's not much difference between these sorts, but for larger lists, there is a clear advantage to merge sort.

Sorting and Searching

- Recall that if we wanted to use binary search, the array must be sorted.
  - What if we sort the array first using merge sort?
    - Merge sort O(n log n) (worst case)
    - Binary search O(log n) (worst case)
    - Total time: (worst case) O(n log n) + O(log n) = O(n log n)

Comparing Big O Functions