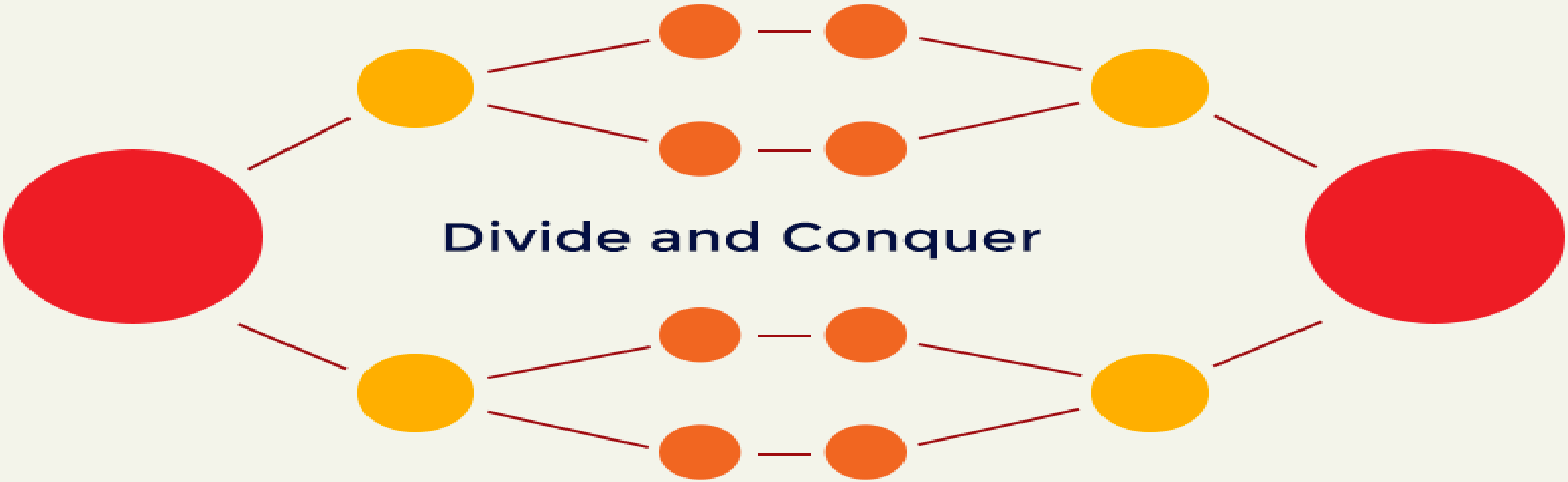
DIVIDE / CONQUER



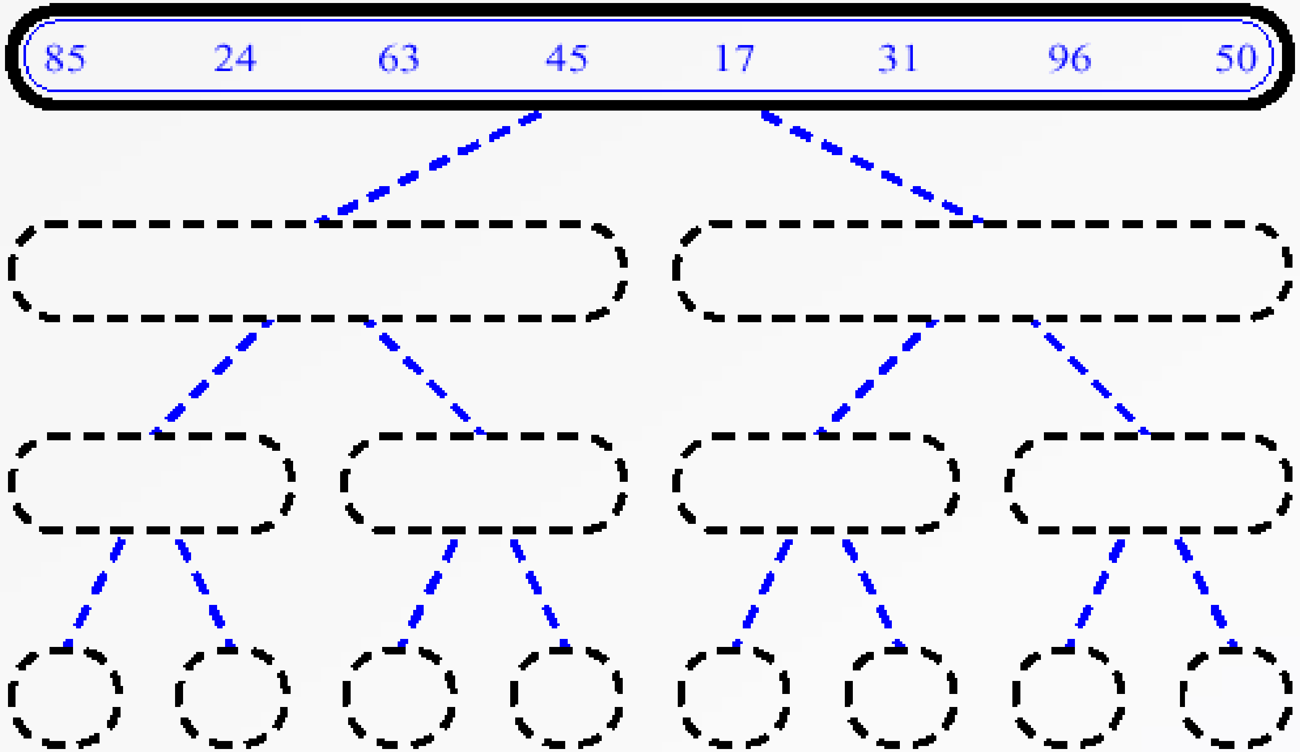
DIVIDE AND CONQUER

– **Divide** the problem into sub-problems that are similar to the original but smaller in size.

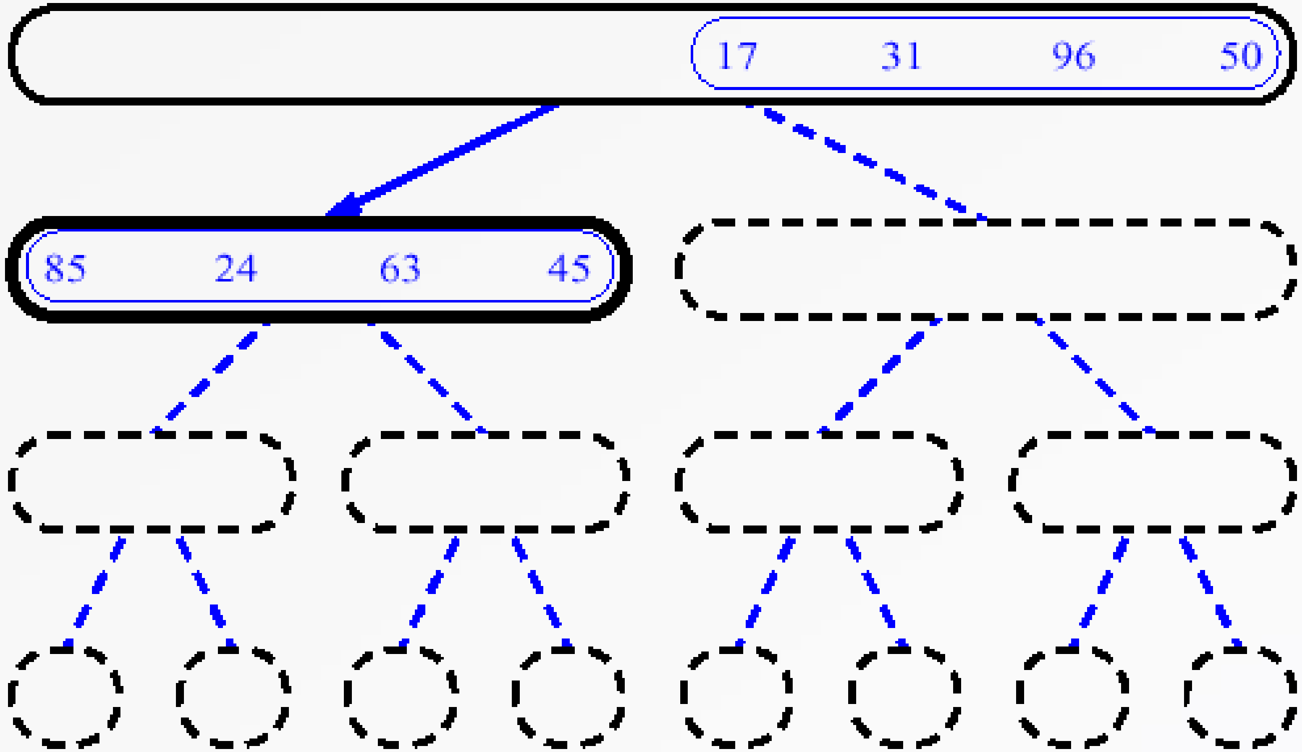
– **Conquer** the sub-problems by solving them recursively. If they are small enough, just solve them in a straightforward manner.

– **Combine** the solutions to create a solution to the original problem

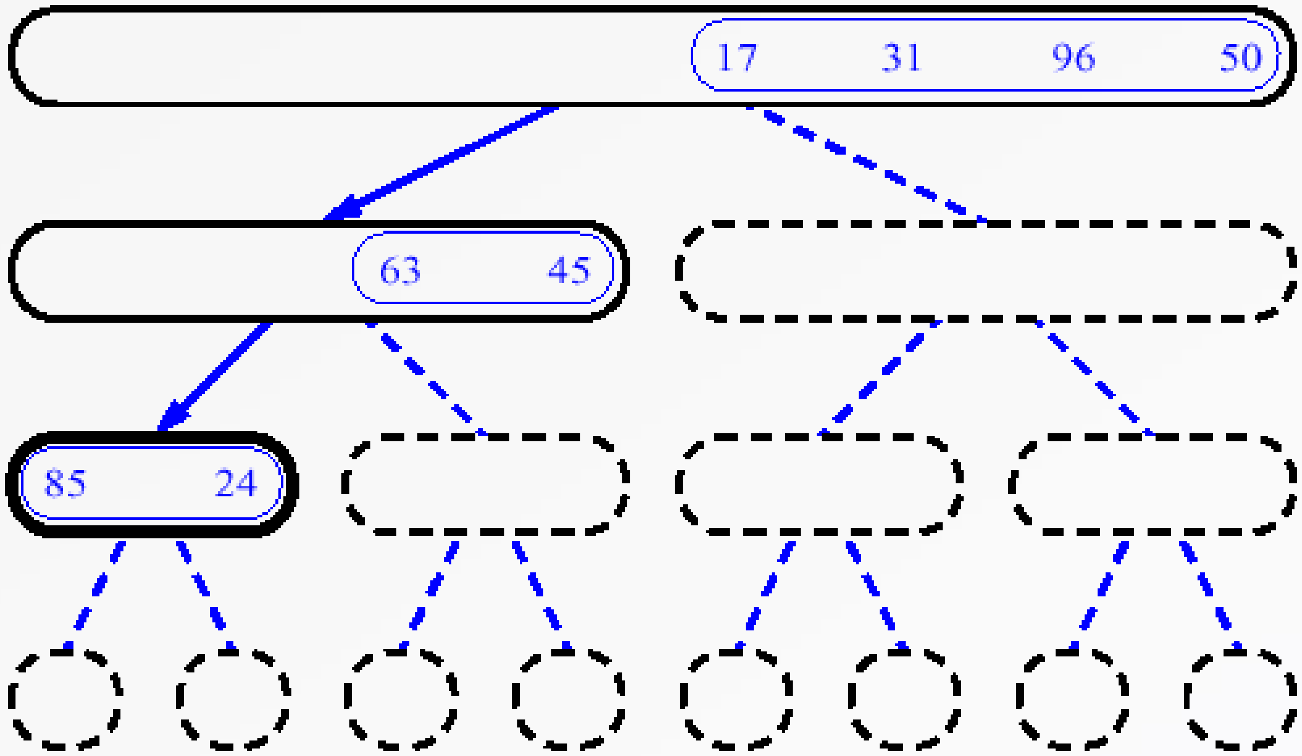
MERGESORT



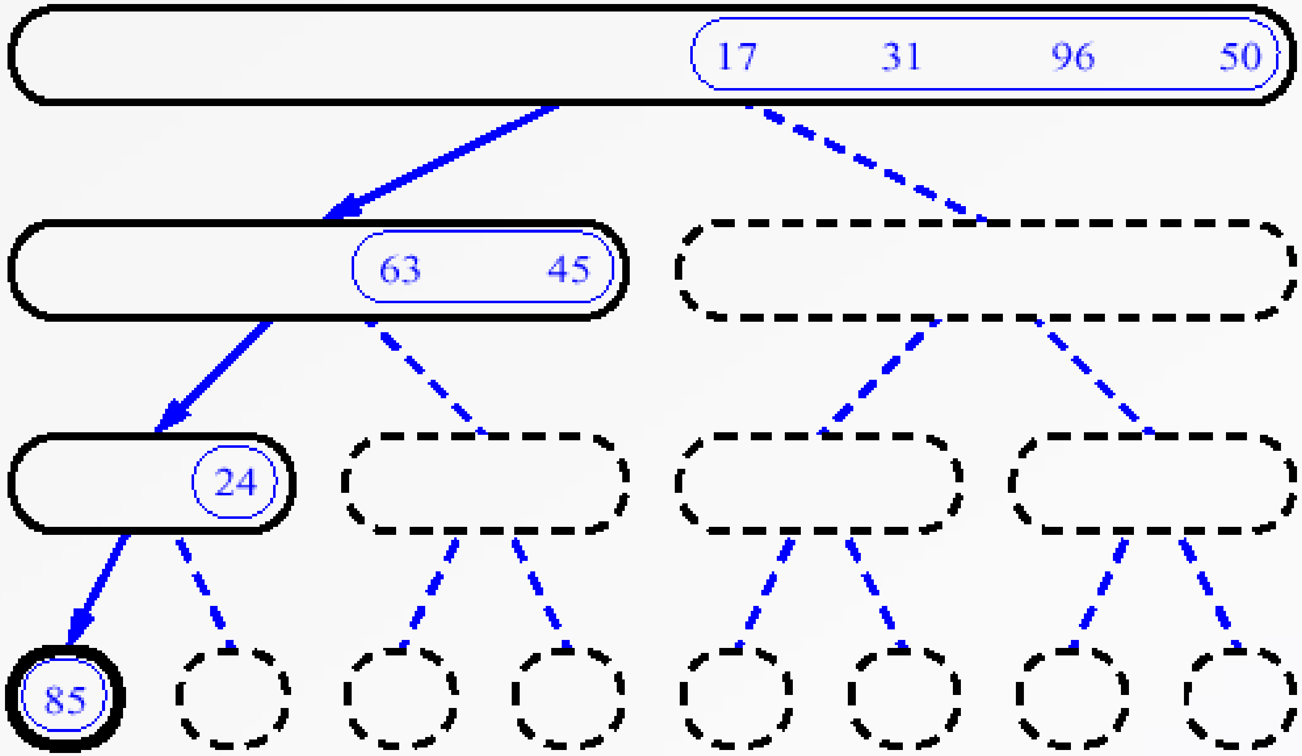
CONT…



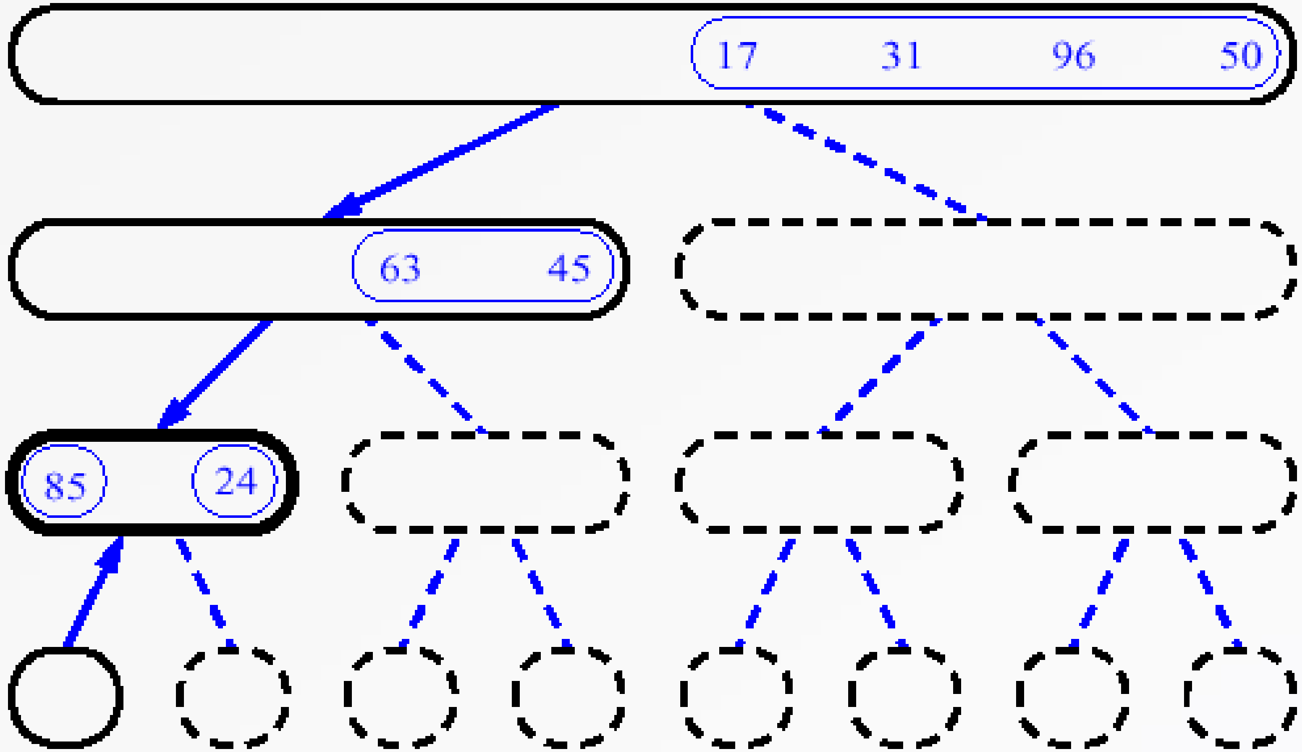
CONT…



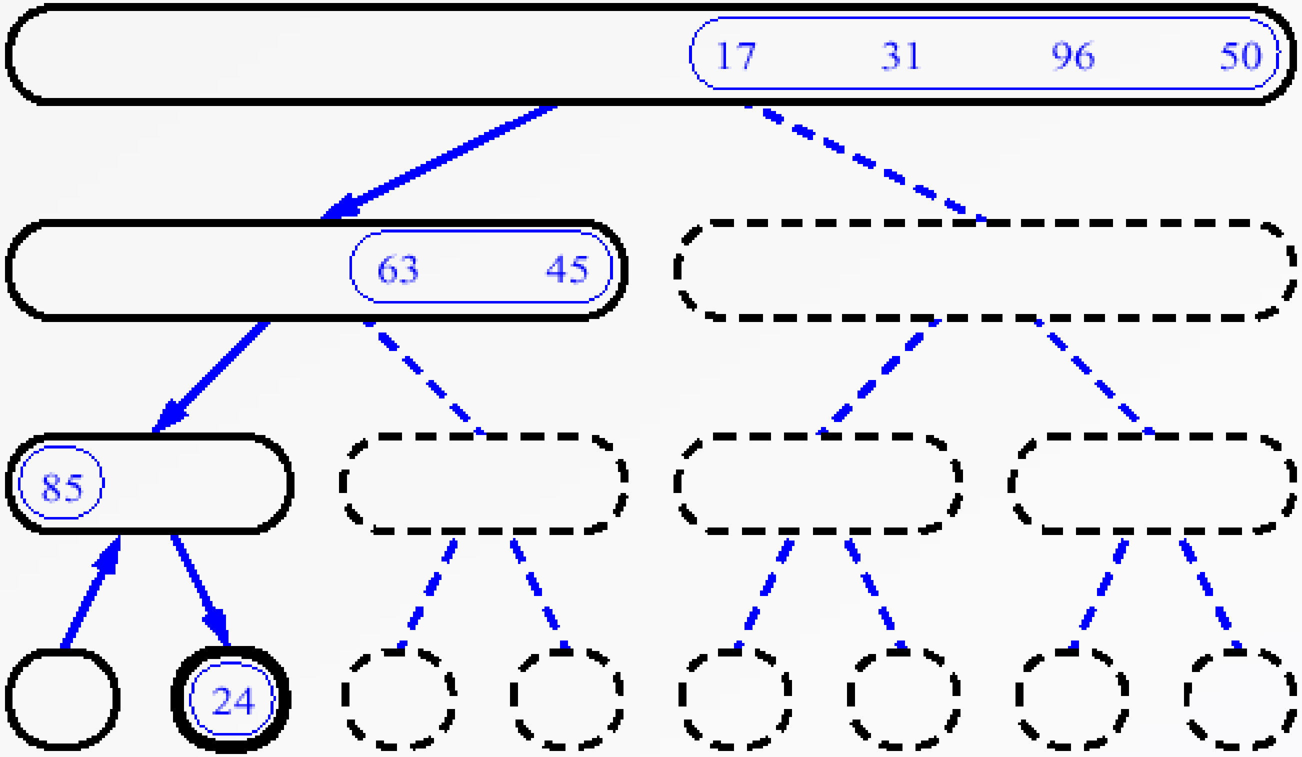
CONT…



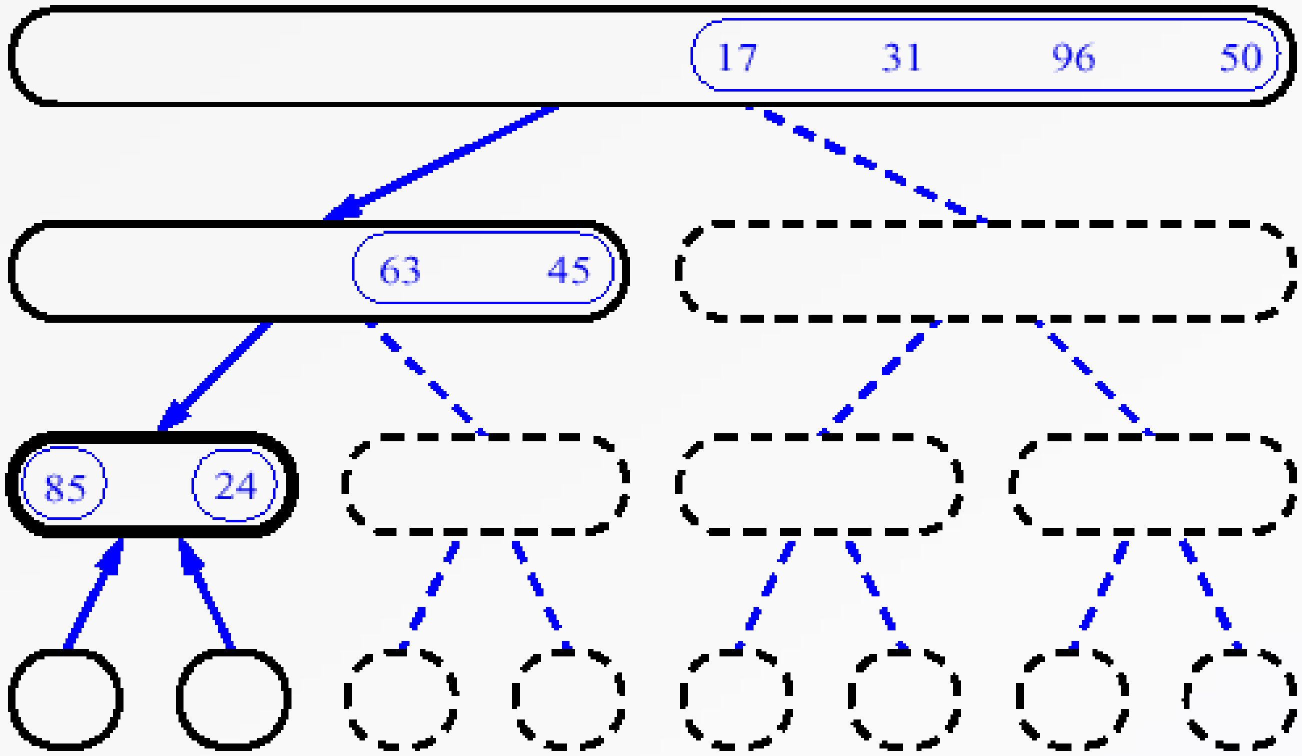
CONT…



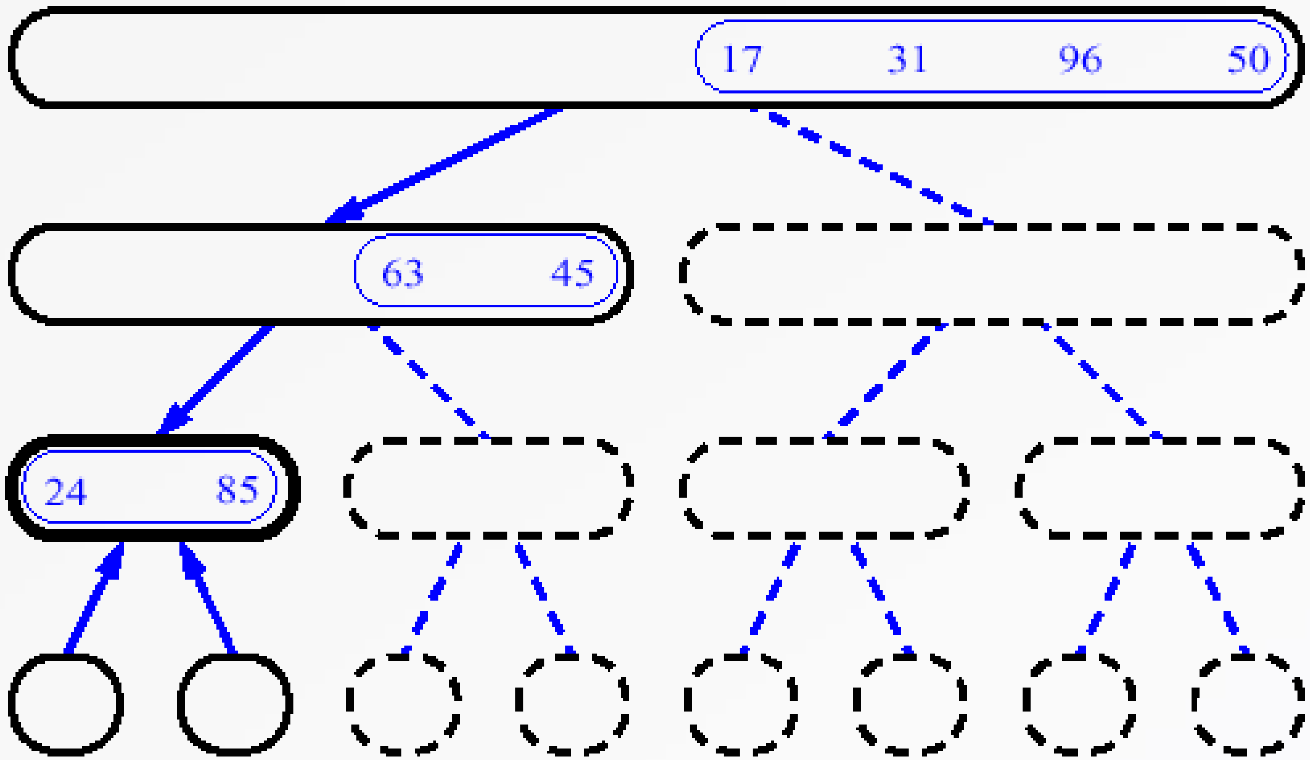
CONT…



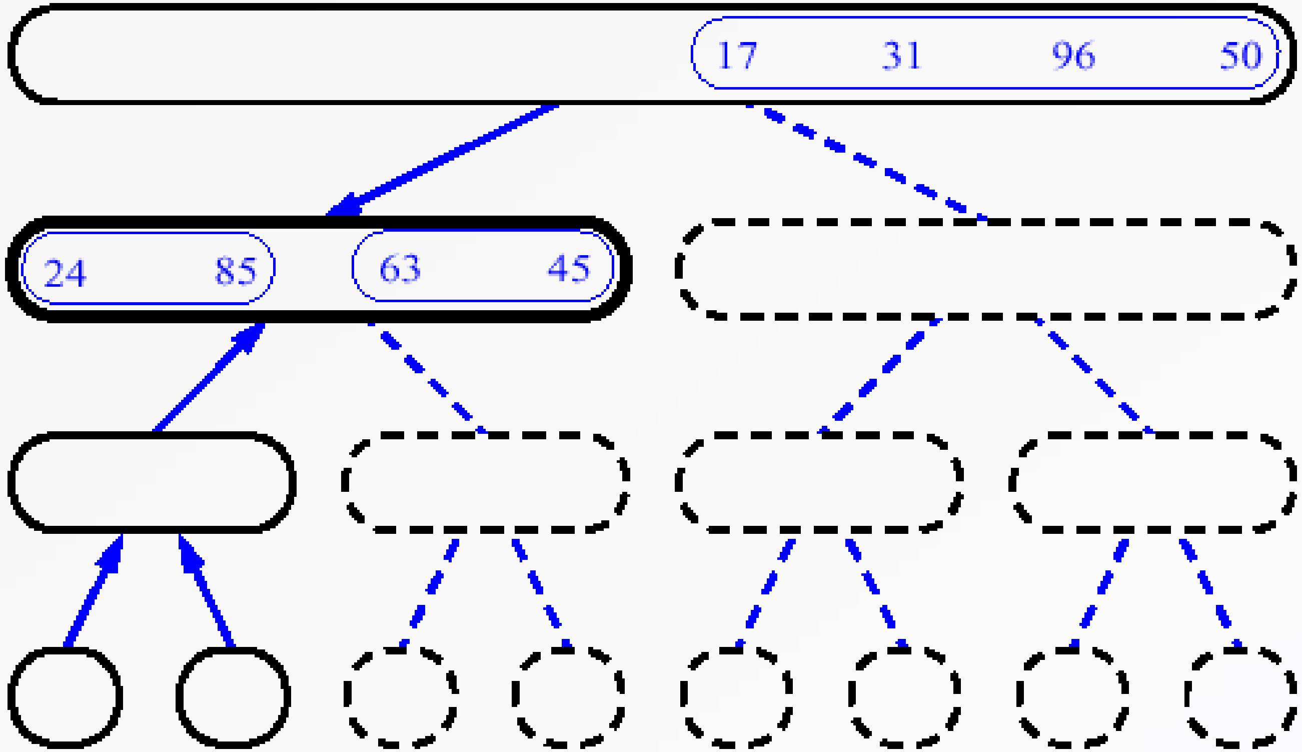
CONT…



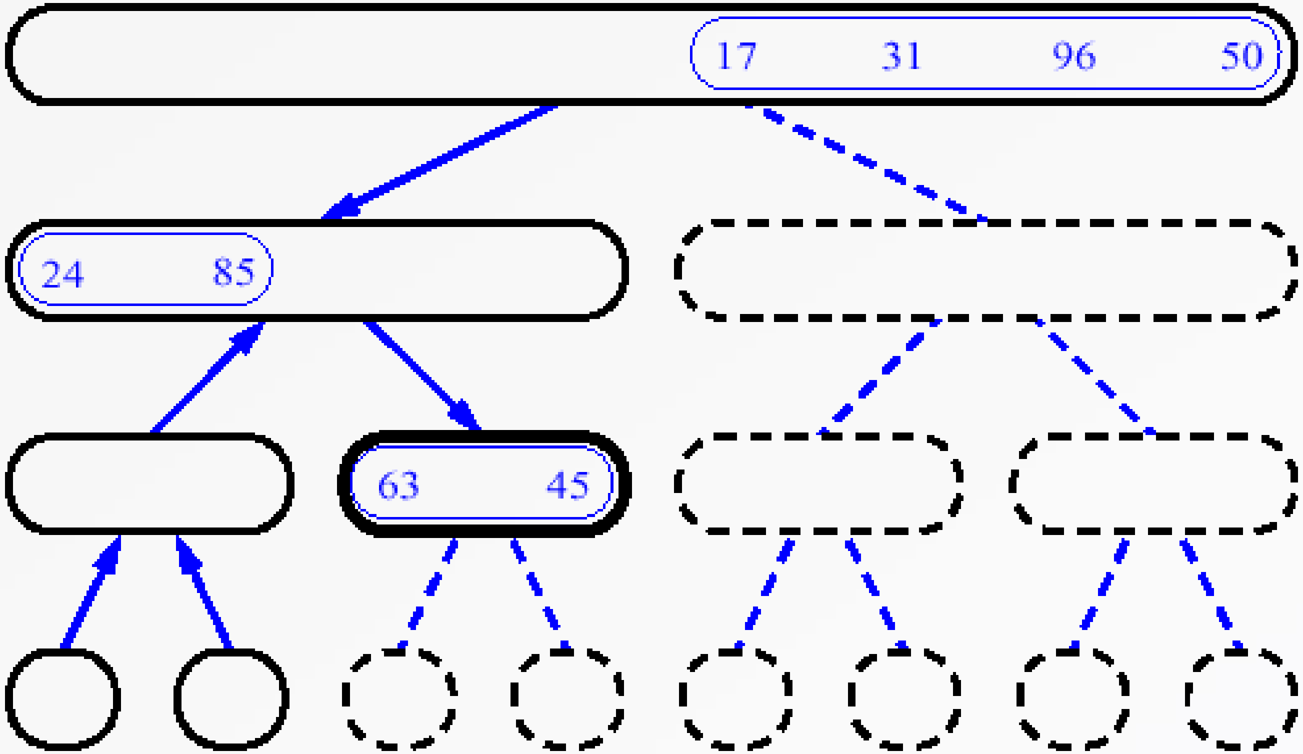
CONT…



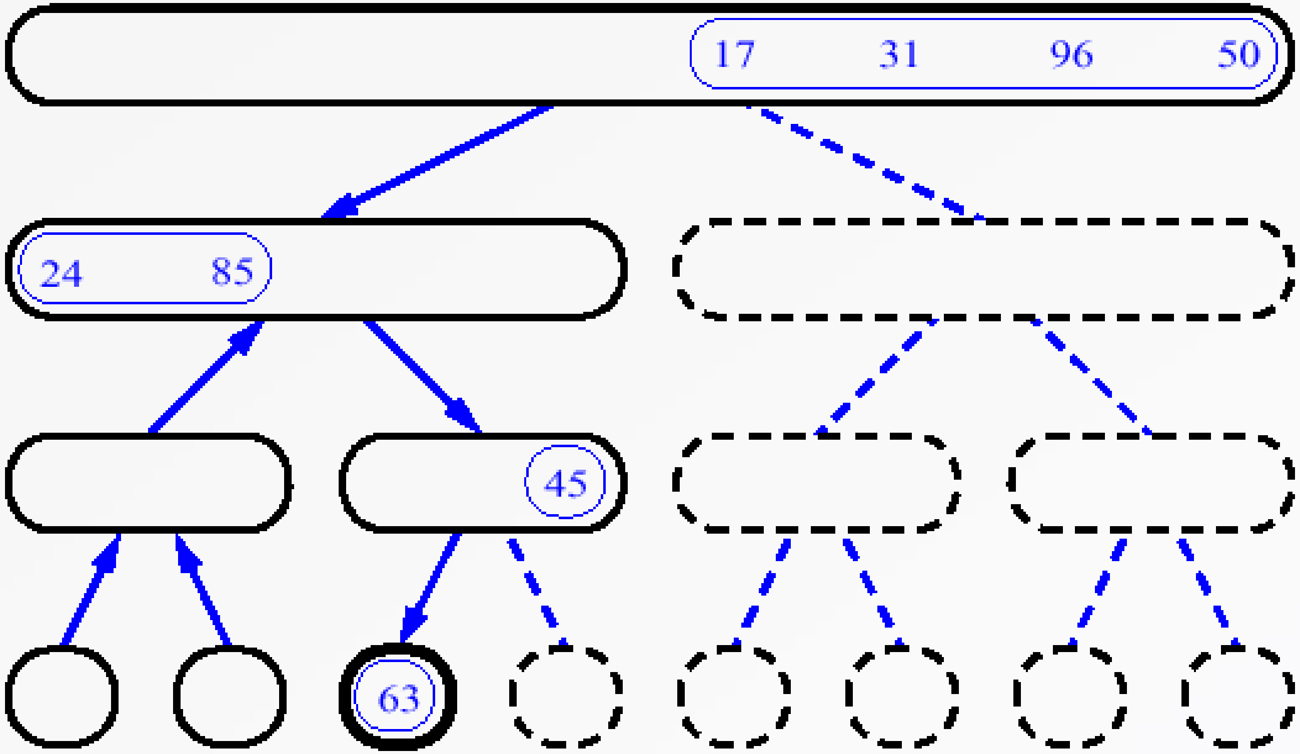
CONT…



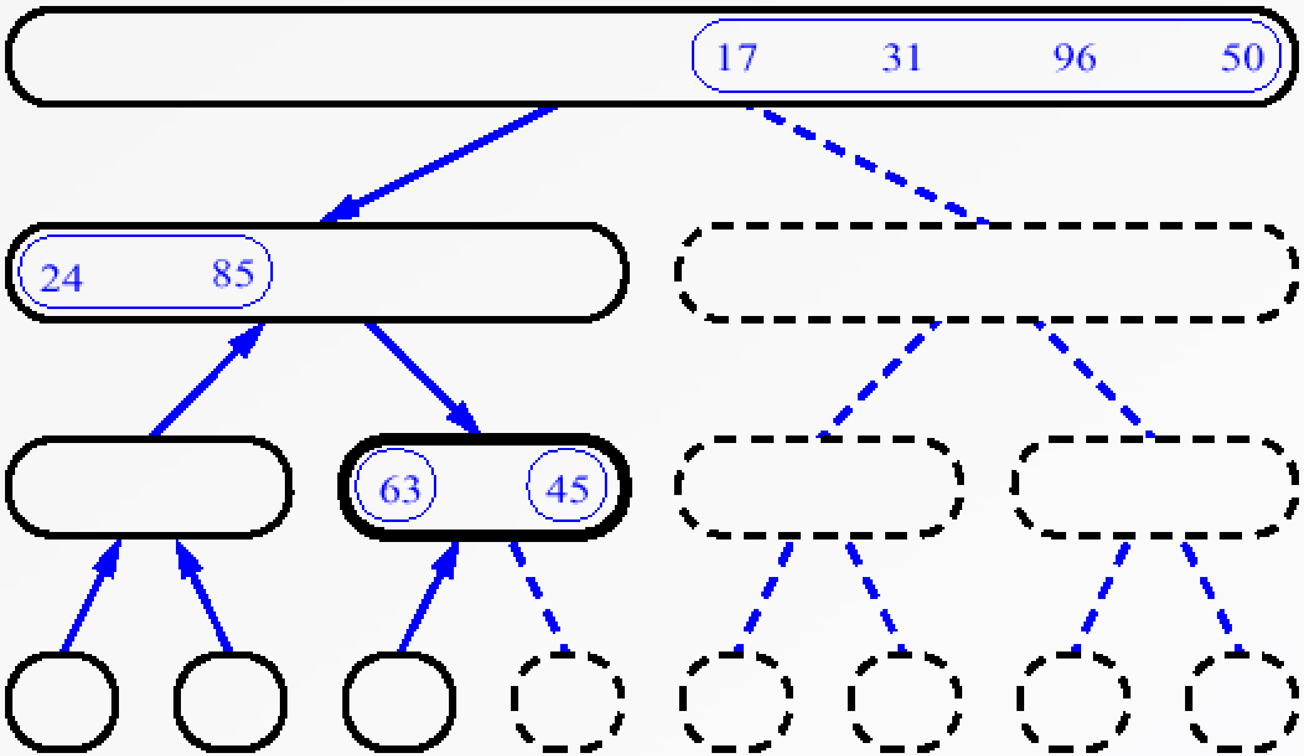
CONT…



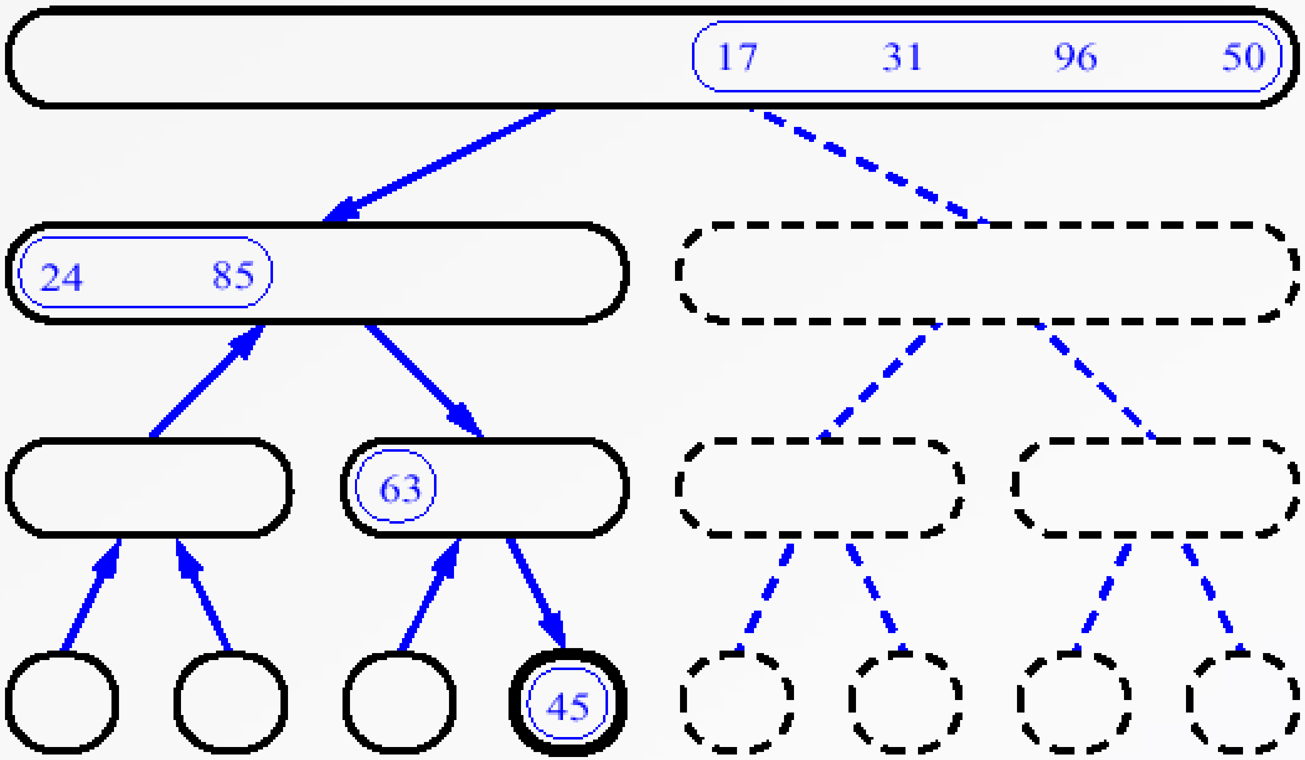
CONT…



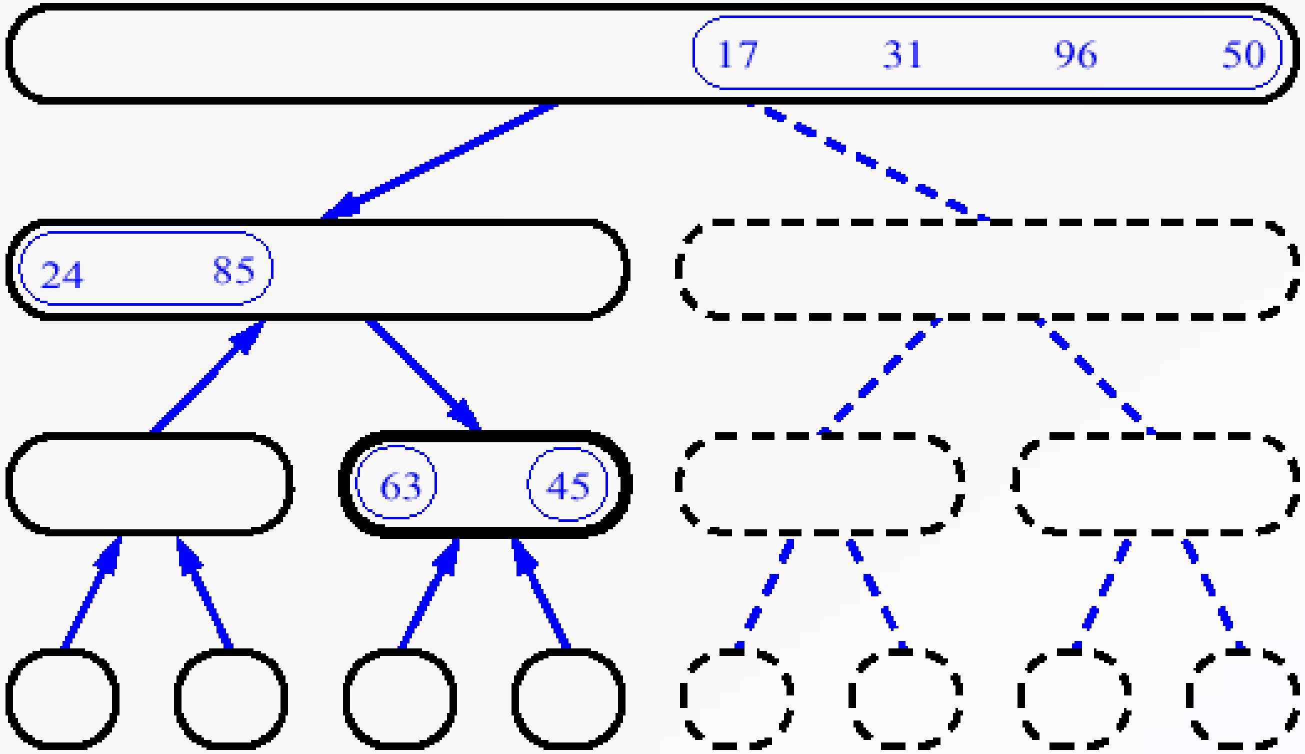
CONT…



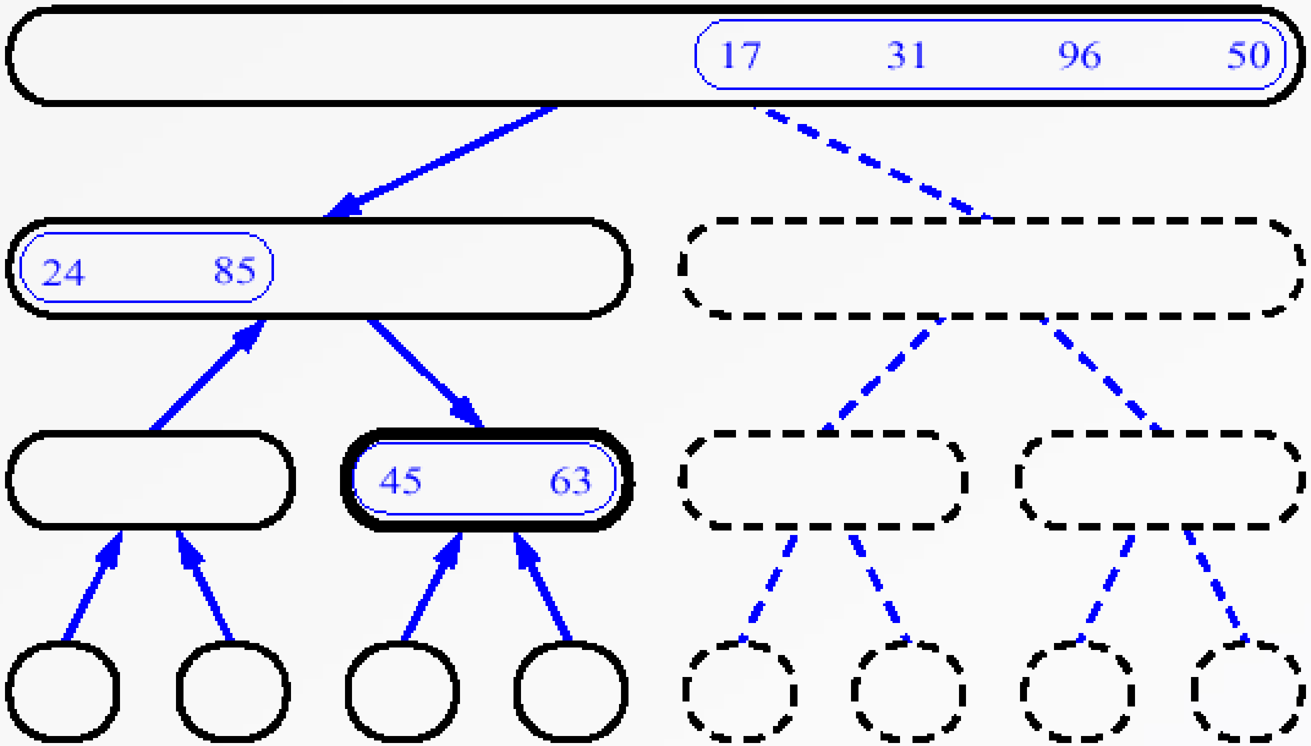
CONT…



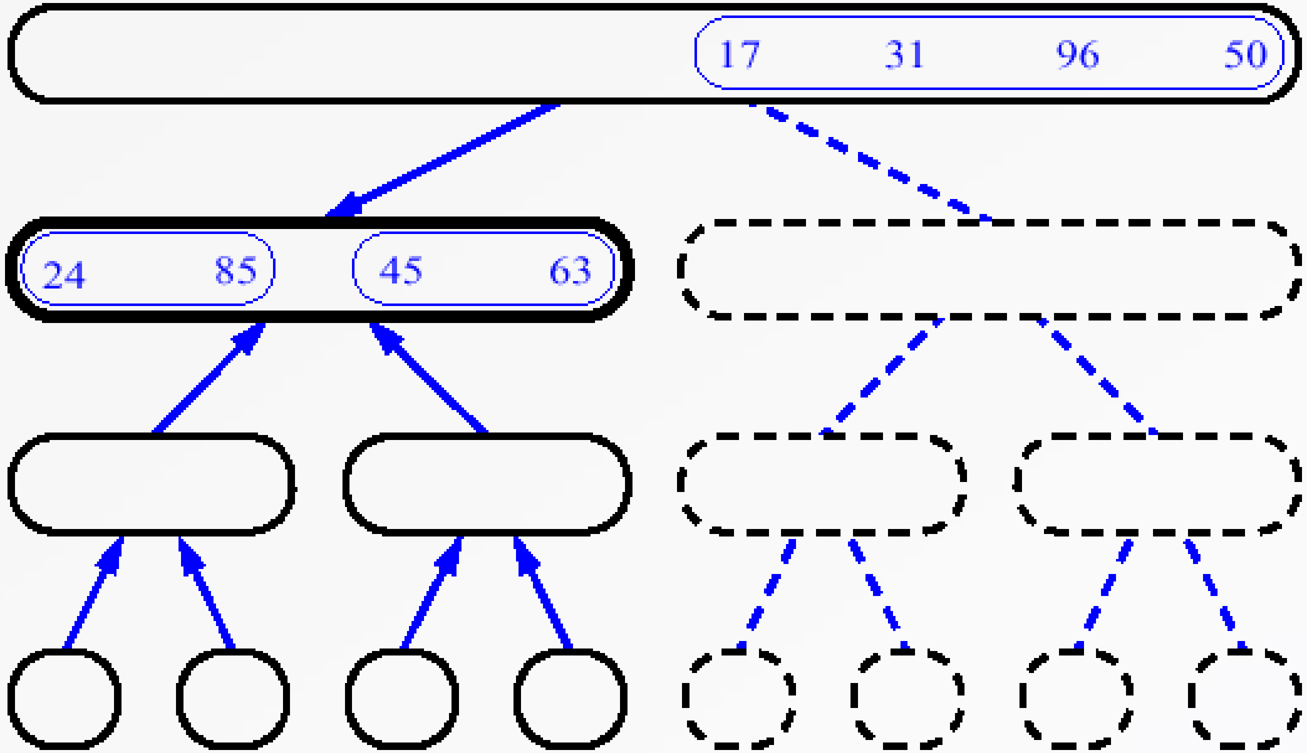
CONT…



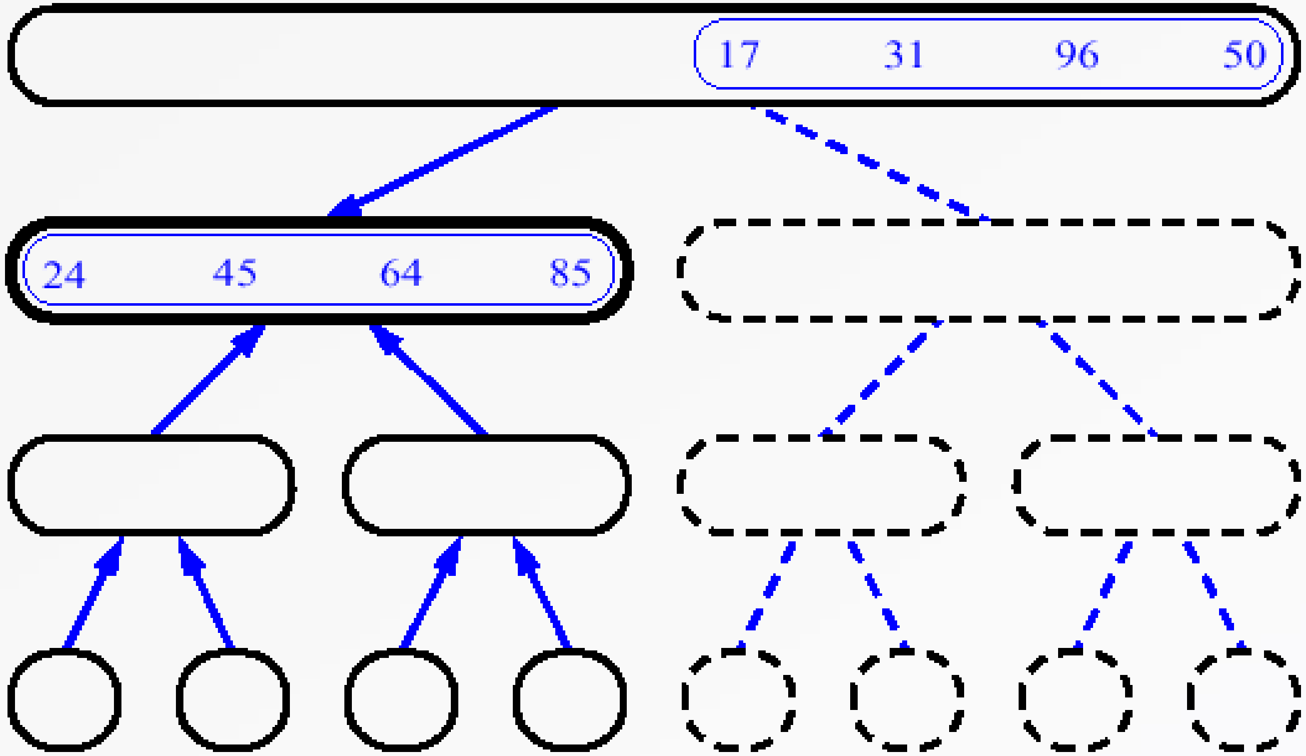
CONT…



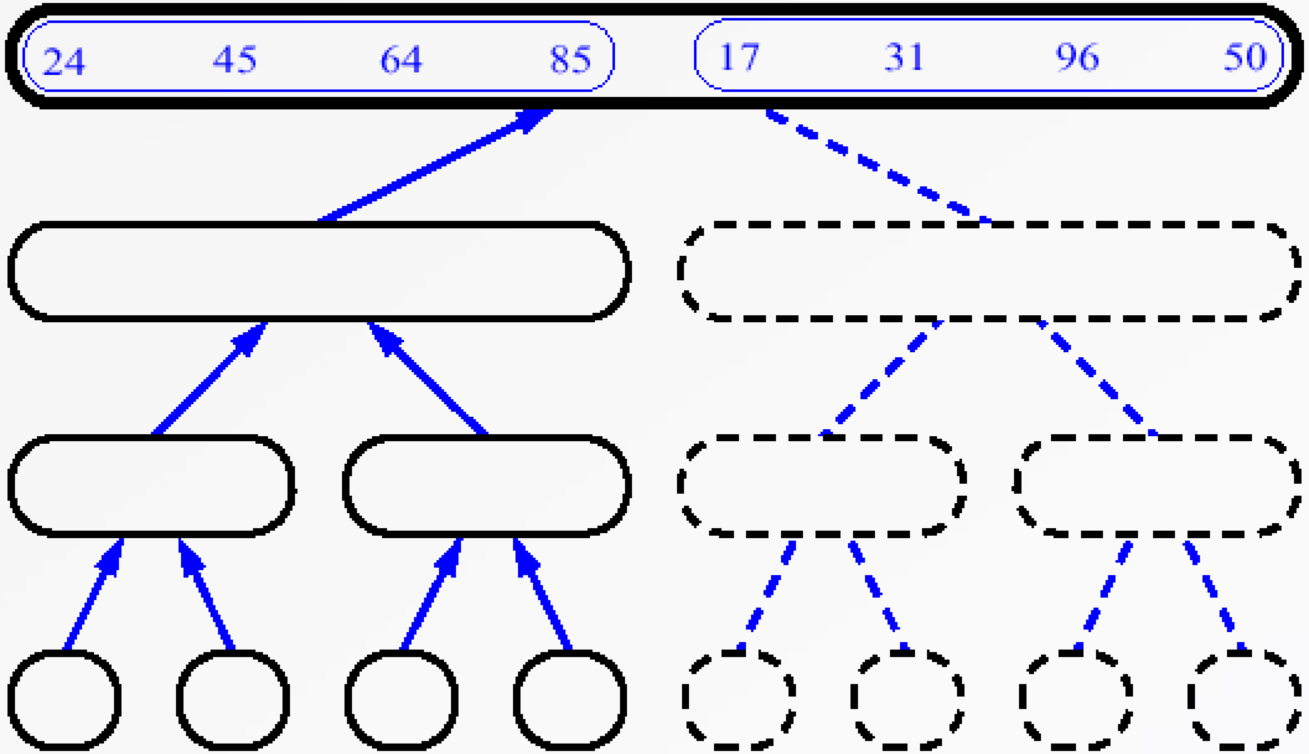
CONT…



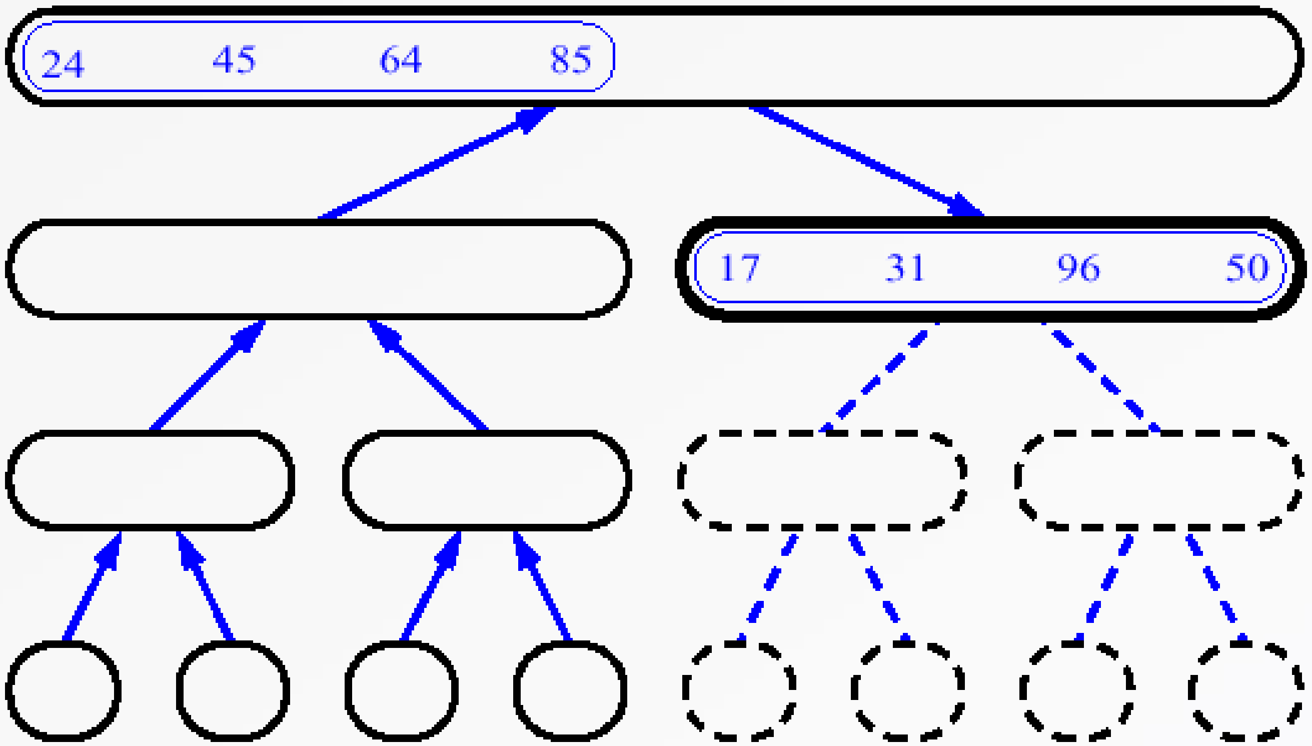
CONT…



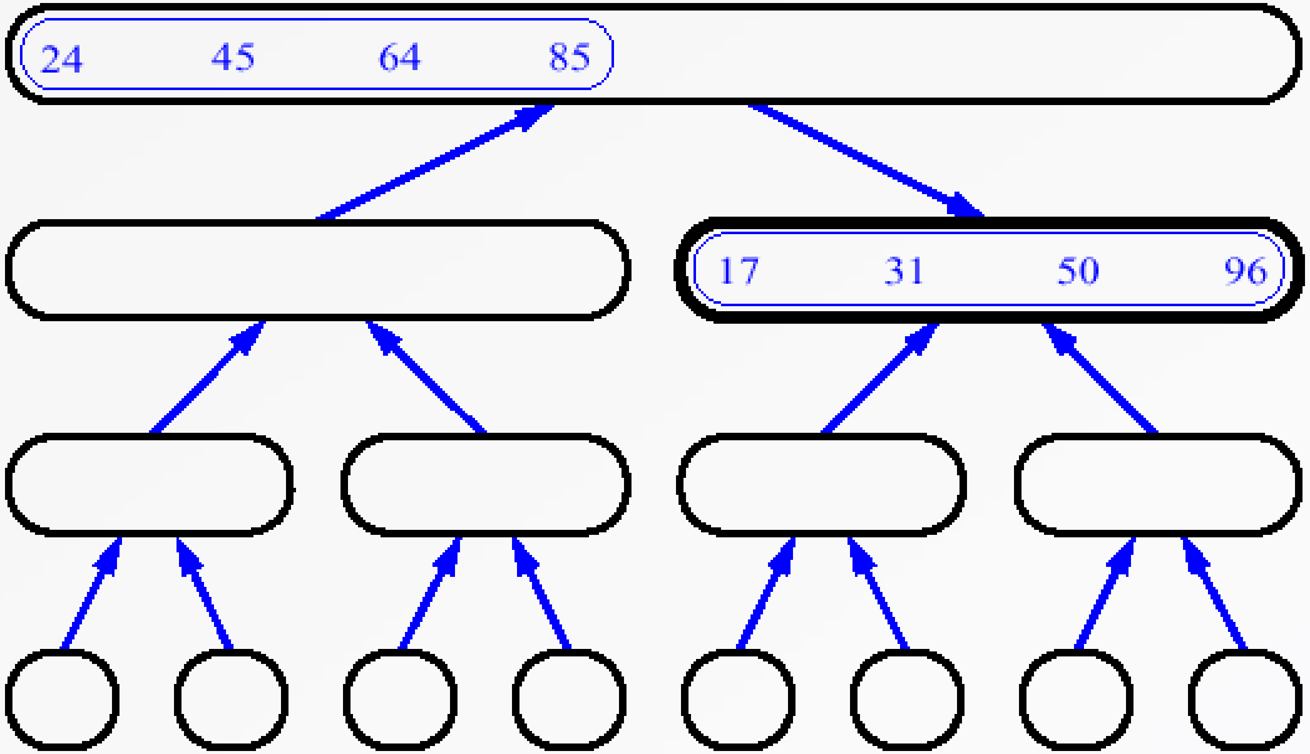
CONT…



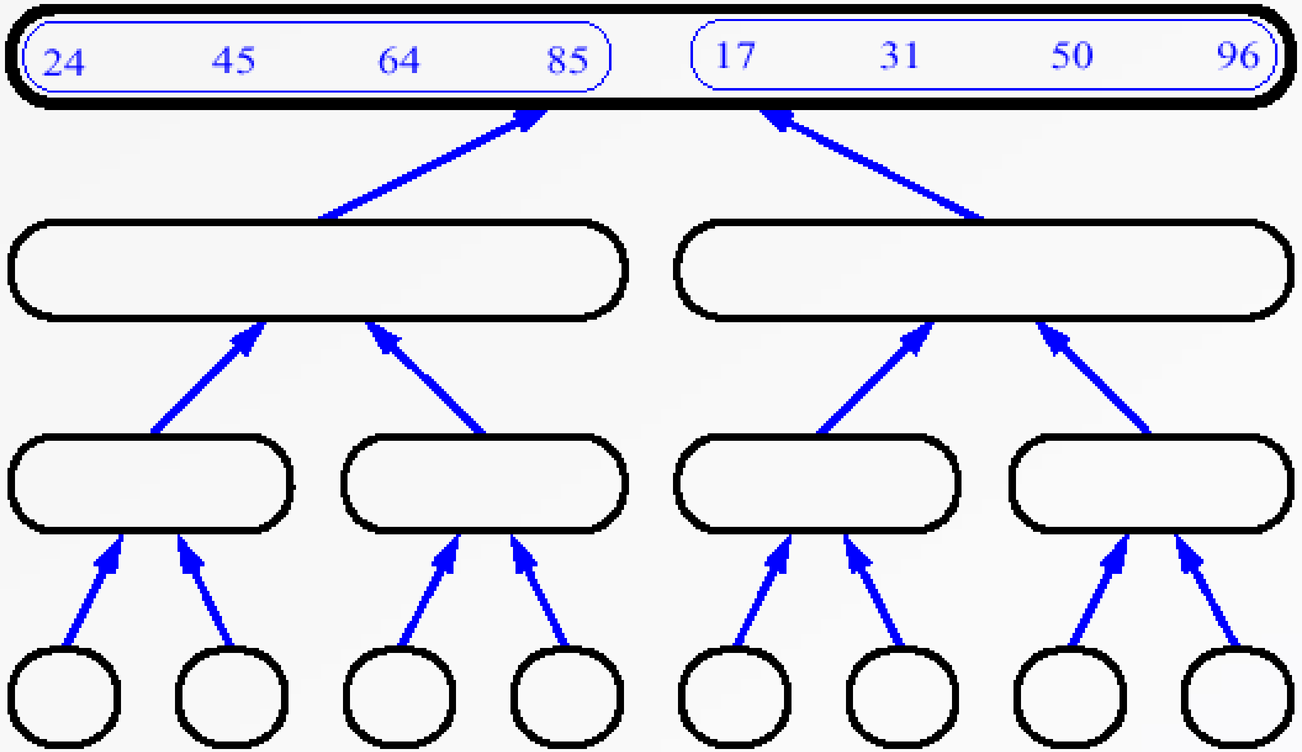
CONT…



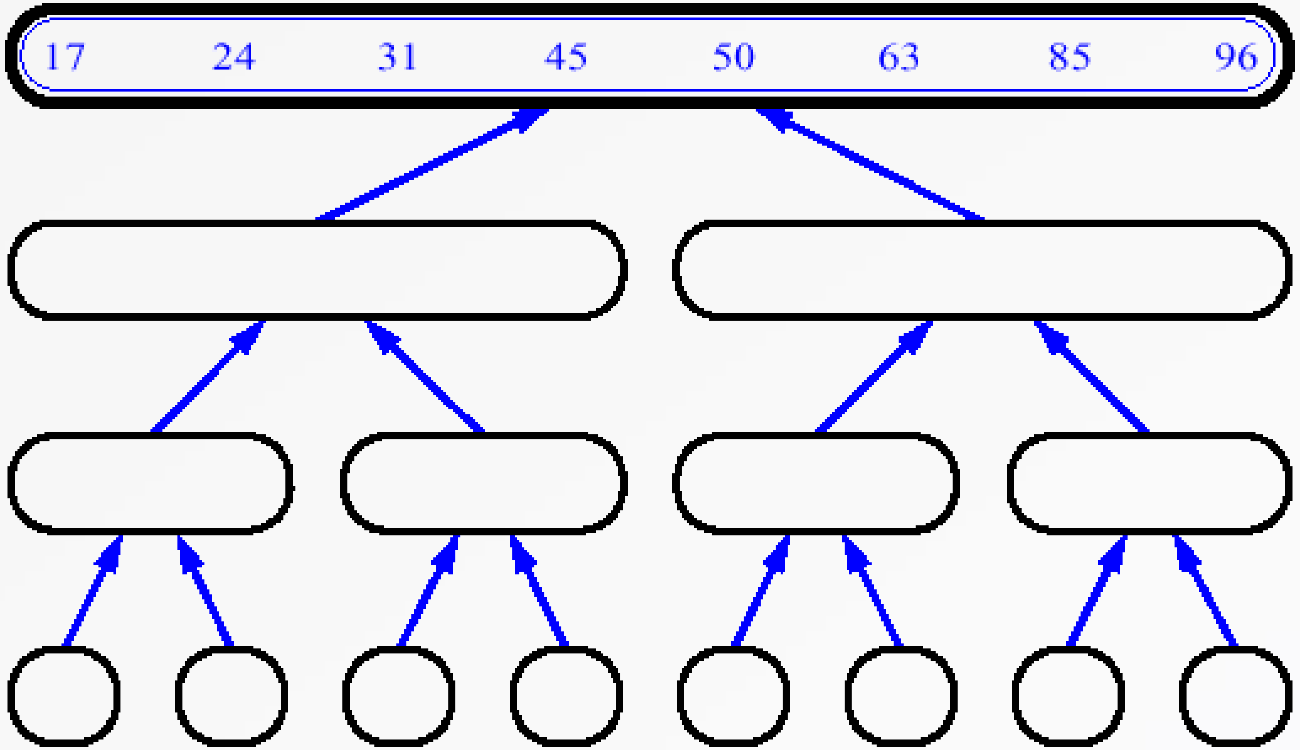
CONT…



CONT…



CONT…



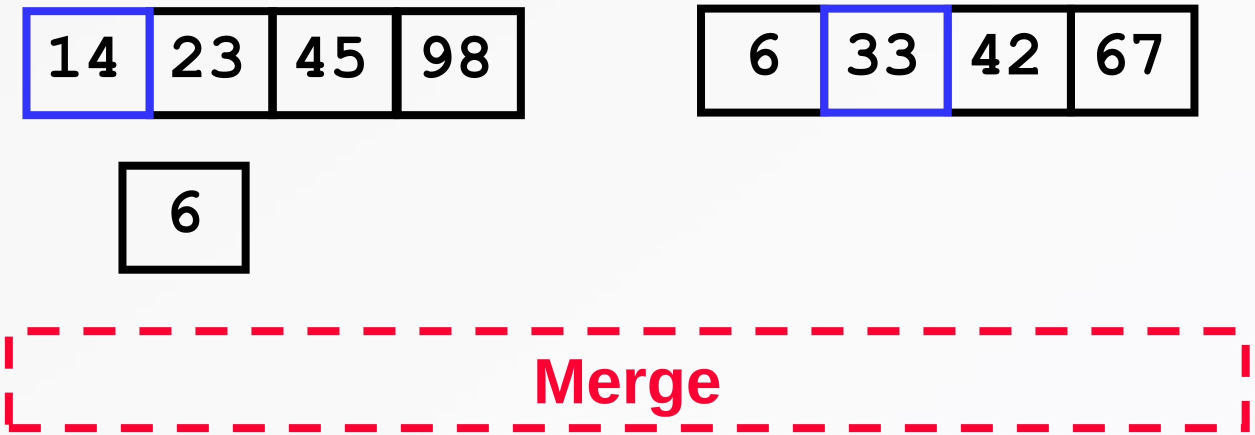
14	23	45	98

6	33	42	67

14	23	45	98

6	33	42	67

Merge

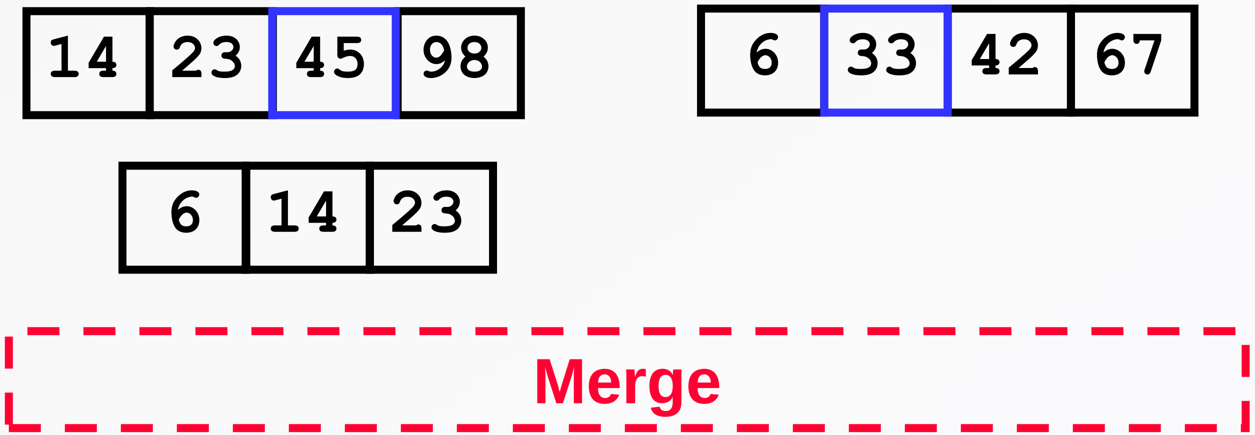

14 23 45 98
6 33 42 67
6
Merge

14	23	45	98

6	33	42	67

6	14

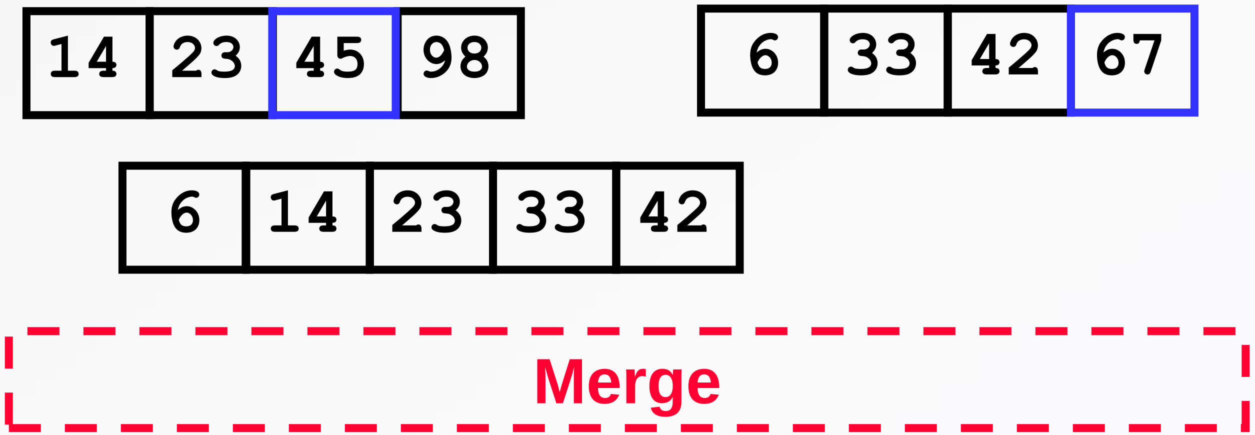
Merge


14 23 45 98
6 33 42 67
6 14 23
Merge

14	23	45	98		6	33	42	67

6	14	23	33

Merge


14 23 45 98
6 33 42 67
6 14 23 33 42
Merge

14	23	45	98

6	33	42	67

6	14	23	33	42	45

Merge

14	23	45	98		6	33	42	67

6	14	23	33	42	45	67

Merge

14	23	45	98

6	33	42	67

6	14	23	33	42	45	67	98

Merge

ALGORITHM

```
MergeSort(A[0....n-1])
  if n>1
      Copy A[0.... (n/2) -1] to B[ 0.... (n/2) -1]
      Copy A[(n/2).....n-1]] to C[ 0.... (n/2) -1]
 MergeSort(B[0....n-1])
 MergeSort(C[0....n-1])
 Merge(B,C,A)
```

```
Merge(B [0....p-1], C[0....q-1], A[0....p+q-1])
 i←0, j←0, k←0
while i<p & j<q do
 if B[i] <= C[j]
   A[K] ← B[i] ; i++;
 else
   A[K] ← C[j] ; j++;
K ←K+1 ;

//Copy left over elements
if i==p
 copy C[j......q-1] to A[k..... p+q-1]
else
 copy B[j......p-1] to A[k..... p+q+1]
```

8 3 2 9 7 1 5 4

MergeSort(A[0....n-1]) → T(n)

if n>1

Copy A[0.... (n/2) -1] to B[0.... (n/2) -1]

Copy A[0.... (n/2) -1] to C[0.... (n/2) -1]

MergeSort(B[0....n-1]) → T(n/2)

MergeSort(C[0....n-1]) → T(n/2)

Merge(B,C,A) → T(n)

$$T(n) \le \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T\left(\lceil n/2 \rceil\right)}_{\text{solve left half}} + \underbrace{T\left(\lfloor n/2 \rfloor\right)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

T(n)= 2T(n/2)

+cn

APPROACHES TO SOLVE RECURSION

Approach 1:

1. Intuitive solution to recurrence is to “unroll” the recursion, accounting for the running time of first few levels.

2. Identify a pattern that can be continued as the recursion expands.

3. Sum the running times over all levels of the recursion and thereby arrives at a total running time.

Step1: Analyze the first few levels.

- 1st level of recursion → Single problem of size n → O(n)
- 2nd level of recursion → 2 problems each of size n/2 → O(n/2)
- 3rd level of recursion → 4 problems each of size n/4 → O(n/4)

Step 2: Identifying the pattern.

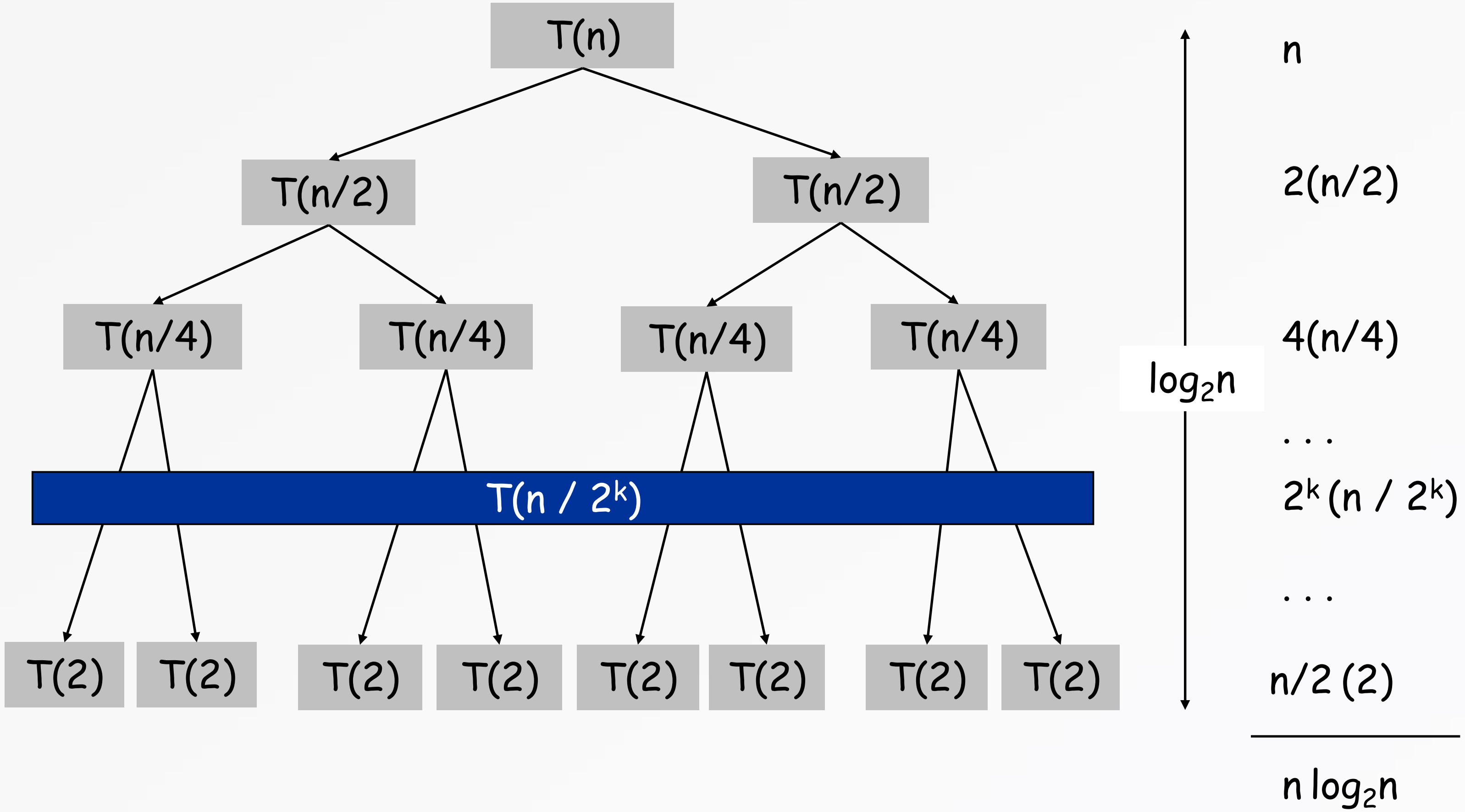
- At level *j* of the recursion, the number of subproblems are now a total of $\mathbf{2^j}$
- Each problem has shrunk in size by factor of 2 “j” time → $n/\mathbf{2^j}$

Step 3: Summing overall levels of recursion.

- The number of times the input must be halved to reduce the size of n to 1 is **log n**
- There are totally “n” levels of recursion → **O(n logn)**

ANALYSIS

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$



SUBSTITUTING A SOLUTION INTO THE MERGESORT RECURSION

$T(n) = O(n \log n)$

$T(n) = \text{c. } n \log n$

$T(n/2) = \text{c. } n/2 \log n/2$

$T(n) = 2T\ (n/2) + O(n)$

$T(n) = 2T\ (n/2) + cn$

$T(n) = 2.\ \mathbf{c.\ n/2 \log n/2} + cn$

$T(n) = cn.\ [\log n - 1] + cn$

$T(n) = cn.\log n - cn + cn$

$T(n) = cn.\log n$

$\boldsymbol{T(n) = O(n\log n)}$

Analysis

- Basic operation - key comparison.
- Best Case, Worst Case, Average Case exists?
  - Execution does not depend on the order of the data
  - Best case and average case runtime are the same as worst case runtime.
- Worst case:
  - During key comparison, neither of the two arrays becomes empty before the other one contains just one element

Analysis – Worst Case

- Assuming for simplicity that total number of elements **n** is a power of 2, the recurrence relation for the number of key comparisons $C(n)$ is

$$C(n) = 2C(n/2) + C_{merge}(n) \quad \text{for } n > 1, \quad C(1) = 0.$$

- $C_{merge}(n)$ - the number of key comparisons performed during the merging stage.
- At each step, exactly one comparison is made, total comparisons are (n-1)

$$C_{worst}(n) = 2C_{worst}(n/2) + n - 1 \quad \text{for } n > 1, \quad C_{worst}(1) = 0.$$

Analysis

$$C_{worst}(n) = 2C_{worst}(n/2) + n - 1 \quad \text{for } n > 1, \quad C_{worst}(1) = 0.$$

- Here a = 2, b = 2, f (n) = n-1 = Θ (n) => d = 1.
- Therefore $2 = 2^1$, case 2 holds in the master theorem
- $\boldsymbol{C_{worst}}$ (n) = Θ (n^d log n) = Θ (n^1 log n) = Θ (n log n)
- Therefore **C_{worst}(n) = Θ (n log n)**

Master theorem

It states that, in recurrence equation T(n) = aT(n/b) + f(n),

If f(n)∈ Θ (n^d) where d ≥ 0 then

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d, \\ \Theta(n^d \log_b n) & \text{if } a = b^d, \\ \Theta(n^{\log_b a}) & \text{if } a > b^d. \end{cases}$$

Quick Sort

Extra!

- It is a Divide and Conquer method
- Sorting happens in Divide stage itself.
- C.A.R. Hoare (also known as Tony Hore), prominent British computer scientist invented quicksort.

Quick Sort

- Quicksort divides (or partitions) array according to the value of some pivot element A[s]
- Divide-and-Conquer:
  - If n=1 terminate (every one-element list is already sorted)
  - If n>1, partition elements into two; based on pivot element

$$\underbrace{A[0]\dots A[s-1]}_{\text{all are } \le A[s]} \quad A[s] \quad \underbrace{A[s+1]\dots A[n-1]}_{\text{all are } \ge A[s]}$$

Quick Sort

$$\underbrace{A[0] \ldots A[s-1]}_{\text{all are } \leq A[s]} \quad A[s] \quad \underbrace{A[s+1] \ldots A[n-1]}_{\text{all are } \geq A[s]}$$

ALGORITHM *Quicksort*$(A[l..r])$

//Sorts a subarray by quicksort
//Input: Subarray of array $A[0..n-1]$, defined by its left and right
// indices l and r
//Output: Subarray $A[l..r]$ sorted in nondecreasing order

if $l < r$
$s \leftarrow Partition(A[l..r])$ //s is a split position
$Quicksort(A[l..s-1])$
$Quicksort(A[s+1..r])$

How do we partition?

- There are several different strategies for selecting a pivot and partitioning.
- We use the sophisticated method suggested by C.A.R. Hoare, the inventor of quicksort.
- Select the subarray’s first element: **p = A[l].**
- Now scan the subarray from both ends, comparing the subarray’s elements to the pivot.

How do we partition?

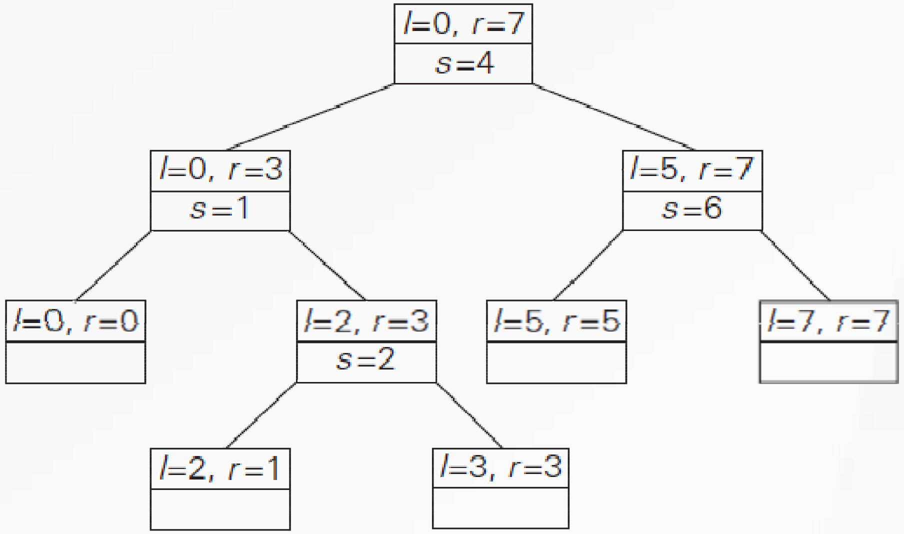
ALGORITHM $HoarePartition(A[l..r])$

//Partitions a subarray by Hoare's algorithm, using the first element
// as a pivot
//Input: Subarray of array $A[0..n-1]$, defined by its left and right
// indices l and r $(l < r)$
//Output: Partition of $A[l..r]$, with the split position returned as
// this function's value

$p \leftarrow A[l]$
$i \leftarrow l;\ j \leftarrow r+1$

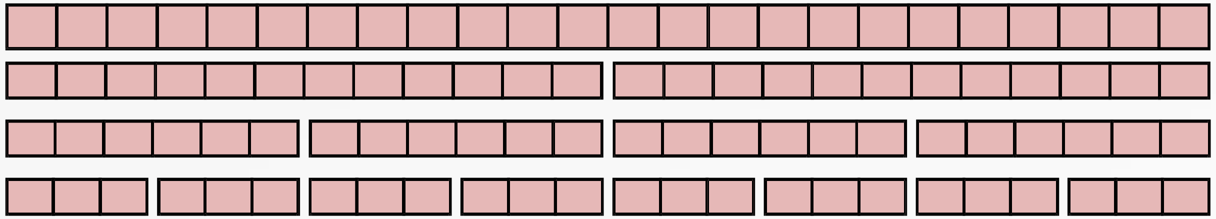
repeat
  repeat $i \leftarrow i+1$ **until** $A[i] \geq p$
  repeat $j \leftarrow j-1$ **until** $A[j] \leq p$
  swap($A[i]$, $A[j]$)
until $i \geq j$
swap($A[i]$, $A[j]$) //undo last swap when $i \geq j$
swap($A[l]$, $A[j]$)
return j

0	1	2	3	4	5	6	7
5	i 3	1	9	8	2	4	j 7
5	3	1	i 9	8	2	j 4	7
5	3	1	i 4	8	2	j 9	7
5	3	1	4	i 8	j 2	9	7
5	3	1	4	i 2	j 8	9	7
5	3	1	4	j 2	i 8	9	7
2	3	1	4	5	8	9	7
2	i 3	1	j 4				
2	i 3	j 1	4				
2	i 1	j 3	4				
2	j 1	i 3	4				
1	2	3	4				
1							
		3	i j 4				
		j 3	i 4				
			4				
					8	i 9	j 7
					8	i 7	j 9
					8	j 7	i 9
					7	8	9
					7		
							9



Analysis

- Basic Operation : Key Comparison
- Best case exists
  - all the splits happen in the middle of subarrays,
  - So the depth of the recursion in $\log_2 n$



$$C_{best}(n) = 2C_{best}(n/2) + n \quad \text{for } n > 1, \quad C_{best}(1) = 0.$$

– As per Master Theorem, $C_{best}(n) \in \Theta(n \log_2 n)$;

Analysis

- Worst Case
  - Splits will be skewed to the extreme
  - This happens if the input is already sorted
- In the worst case, partitioning always divides the size n array into these three parts:
  - A length one part, containing the pivot itself
  - A length zero part, and
  - A length n-1 part, containing everything else
- Recurring on the length n-1 part requires (in the worst case) recurring to depthn-1

Analysis

- Worst Case

Analysis

Worst Case

- if A[0..n − 1] is a strictly increasing array and we use A[0] as the pivot,
  - the left-to-right scan will stop on A[1] while the right-to- left scan will go all the way to reach A[0], indicating the split at position 0
  - n + 1 comparisons required

Total comparisons

$$C_{worst}(n) = (n+1) + n + \cdots + 3 = \frac{(n+1)(n+2)}{2} - 3 \in \Theta(n^2).$$

Analysis

Average Case

- Let $C_{avg}(n)$ be the average number of key comparisons made by quicksort on a randomly ordered array of size n.
- A partition can happen in any position s $(0 \le s \le n-1)$
- n+1 comparisons are required for partition.
- After the partition, the left and right subarrays will have s and $n - 1 - s$ elements, respectively.

Analysis

Average Case

- Assuming that the partition split can happen in each position s with the same probability 1/n, we get

$$C_{avg}(n) = \frac{1}{n}\sum_{s=0}^{n-1}[(n+1) + C_{avg}(s) + C_{avg}(n-1-s)] \quad \text{for } n > 1,$$

$$C_{avg}(0) = 0, \quad C_{avg}(1) = 0.$$

$$C_{avg}(n) \approx 2n \ln n \approx 1.39n \log_2 n.$$