Divide and Conquer

Divide and Conquer

- Break problem into several subproblems that are similar to the original problem but smaller in size
- Solve the subproblems recursively
- Combine these solutions to create a solution to the original problem
- Most of the time written recursively

Decrease and Conquer

- Similar to divide and conquer
- Instead of partitioning a problem into multiple subproblems of smaller size, we use some technique to reduce our problem into a single problem that is smaller than the original.
- Solution of smaller size problem is solution of original problem

Binary Search

- **Input:** An array of elements, A and a search element S
- **Output:** Index of S in A if present and –1 otherwise
- **Assumption:** All elements in A are positive
- **Pre-requisite:** Elements in A are in sorted order

Illustration

A

1	2	3	4	5	6
21	34	37	41	45	50

S 37

Illustration

A

1	2	3	4	5	6
21	34	37	41	45	50

S 50

Illustration

A

1	2	3	4	5	6
21	34	37	41	45	50

S 41

Illustration

A

1	2	3	4	5	6
21	34	37	41	45	50

S 43

Algorithm

```
int Binary_Search(A,n,Key):
    l = 1; h = n
    while (l<=h):
        mid = (l+h)/2
        if (key==A[mid])
            return mid
        if (key<A[mid])
            h = mid–1
        else
            l = mid+1
    return 0
```

Proof of Correctness

Strong Induction is used to prove

Base case: Prove that the proposition holds for $n = 0$, i.e., prove that $P(0)$ is true.

induction hypothesis: Assume that $P(n)$ holds for all n between 0 and k

Inductive step: Prove that $P(k+1)$ is true.

Conclude by strong induction that $P(n)$ holds for all $n \geqslant 0$.

Proof of Correctness

Base case: When $n = 0$, $l = 1$ and $h = 0$, loop of the algorithm do not get executed even once and 0 will be returned

Correct for base case

Proof of Correctness

Induction hypothesis: When the size of the array, $n \leq k$, the algorithm will return correct answer

Inductive step: Prove that when the size of the array, $n=k+1$, the algorithm will return correct answer

Proof of Correctness

Case 1: When $S = A[mid]$, algorithm will return mid

Case 2: When $S < A[mid]$, since the array is sorted, S will be present in the subarray from index 1 to mid−1, as per induction hypothesis the algorithm will return correct answer

Case 3: When $S > A[mid]$, symmetrical case of case 2

Induction or Subsitution Method

$T(n) = \; 1 \qquad\qquad n=1$

$\qquad\quad T(n/2) + 1 \qquad n>1$

$T(n) = T(n/2) + 1$ ——————————(1)

$T(n/2) = T(n/2^2) + 1$ ——————————(2)

Applying (2) in (1)

$T(n) = T(n/2^2) + 2$

Induction or Subsitution Method

$T(n/2^2) = T(n/2^3) + 1$ ——————————(3)

Applying (2) in (1)

$T(n) = T(n/2^3) + 3$

.

.

.

$T(n) = T(n/2^k) + k$

At the kth iteration, there will be only one element => $n = 1$

$n/2^k = 1$

Induction or Subsitution Method

$n = 2^k$ and $k = \log n$

$T(n) = 1 + \log n$

$O(\log n)$ or $\theta(\log n)$

Recurrence Tree Method

Master's Theorem for Divide and Conquer

Consider a recurrence relation of the form:

$T(n) = aT(n/b) + \theta(n^k \log^p n)$

1) If $a>b^k$, then $T(n) = \theta(n^{\log_b a})$

2) If $a = b^k$, then

   a. If $p > -1$, then $T(n) = \theta(n^{\log_b a} \log^{p+1} n)$

   b. If $p = -1$, then $T(n) = \theta(n^{\log_b a} \log \log n)$

   c. If $p<-1$ then $T(n) = \theta(n^{\log_b a})$

Master's Theorem for Divide and Conquer

3) If $a<b^k$,

a. If $p \geqslant 0$, then $T(n) = \theta(n^k \log^p n)$

b. If $p < 0$, then $T(n) = \theta(n^k)$

Master's Theorem for Binary Search

$T(n) = T(n/2) + 1$

$a = 1, b = 2, k = 0, p = 0$

Case 2 of master theorem $a = b^k$ and $p > -1$

$\theta(n^0 \log^1 n) = \theta(\log n)$

Master's Theorem for Binary Search

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

$a = 2, b = 2, k = 1, p = 0$

Case 2 of master theorem $a = b^k$ and $p > -1$

$\theta(n^1 \log^1 n) = \theta(n \log n)$

Master's Theorem for Binary Search

$T(n) = 4T(n/2) + n^2$

$a = 4, b = 2, k = 2, p = 0$

Case 2 of master theorem $a = b^k$ and $p > -1$

$\theta(n^2 \log^1 n) = \theta(n^2 \log n)$

Master's Theorem for Binary Search

$T(n) = 4T(n/2) + n^2 \log n$

$a = 4, b = 2, k = 2, p = 1$

Case 2 of master theorem $a = b^k$ and $p > -1$

$\theta(n^2 \log^2 n)$

Master's Theorem for Binary Search

$T(n) = 4T(n/2) + n^2 \log^2 n$

$a = 4, b = 2, k = 2, p = 2$

Case 2 of master theorem $a = b^k$ and $p > -1$

$\theta(n^2 \log^3 n)$

Master's Theorem for Binary Search

$T(n) = 8T(n/2) + n^3$

$a = 8, b = 2, k = 3, p = 0$

Case 2 of master theorem $a = b^k$ and $p > -1$

$\theta(n^3 \log n)$

Merge Sort

Closely follows the divide–and–conquer paradigm

Divide: Divide the n–element sequence to be sorted into two subsequences of n/2 elements each

Conquer: Sort the two subsequences recursively using merge sort

Combine: Merge the two sorted subsequences to produce the sorted answer

Merge Sort

- Recursion stops when sequence to be sorted has a length 1
- Since every sequence of length 1 is already in sorted order
- Key operation of merge sort algorithm is merging of two sorted sequences in "combine" step
- Let us use an Auxiliary procedure MERGE to do the combine operation

Merge Sort

- MERGE (A, p, q, r), where A is an array and p, q, and r are indices into the array such that $p \leq q < r$
- Procedure assumes that the subarrays A[p..q] and A[q+1..r] are in sorted order
- It merges them to form a single sorted subarray that replaces the current subarray A[p..r]

Merge in Merge Sort

- MERGE procedure takes time $\theta(n)$, where $n = r - p + 1$ is the total number of elements being merged
- Shall be illustrated with two piles of cards face up on a table
- Each pile is sorted, with the smallest cards on top
- We wish to merge the two piles into a single sorted output pile, which is to be face down on the table

Merge in Merge Sort

Basic Step

(i) Choose the smaller of the two cards on top of the face–up piles, removing it from its pile (which exposes a new top card),

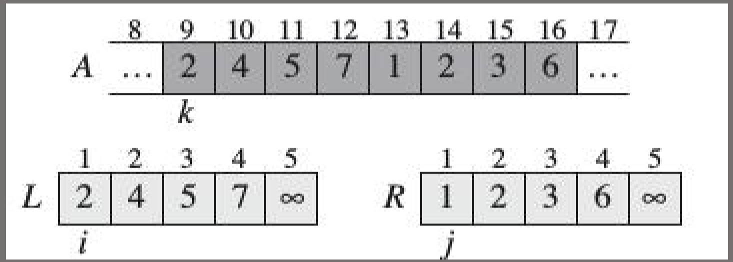
(ii) Place this card face down onto the output pile

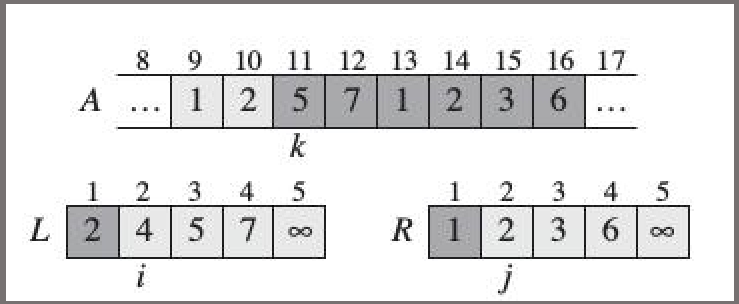
repeat this step until one input pile is empty then take the remaining input pile and place it face down onto the output pile

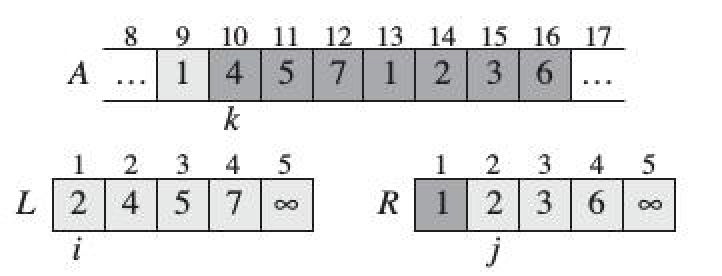
Merge in Merge Sort

- Computationally, each basic step takes constant time, since we are comparing just the two top cards
- Since we perform at most n basic steps, merging takes θ (n) time
- We shall make the algorithm more clear by adding a sentinel card which has a very large value

Illustration of Merge in Merge Sort







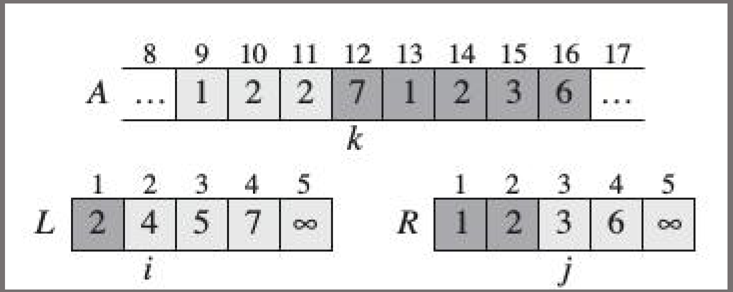
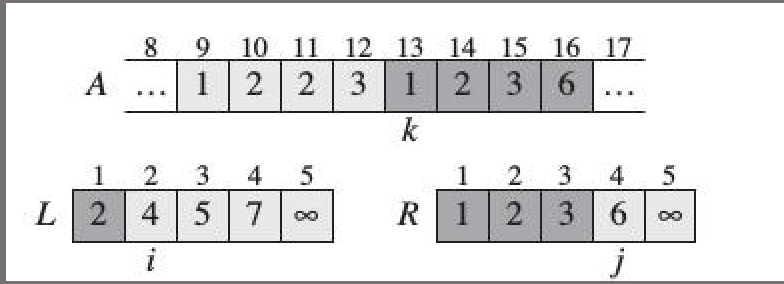
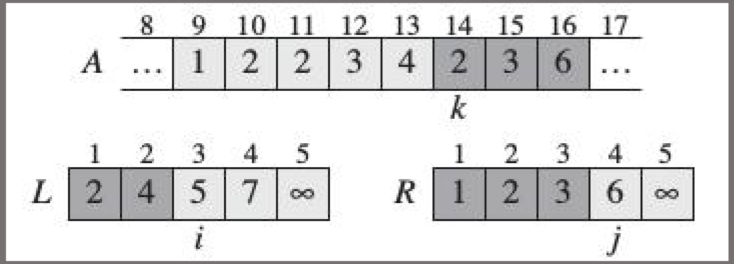
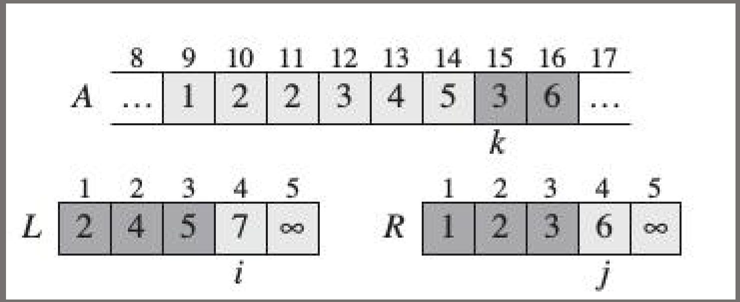
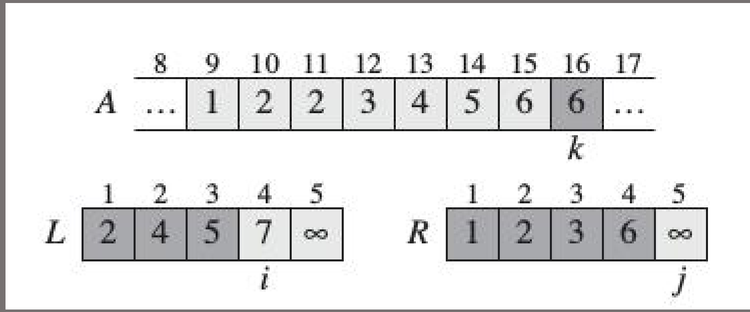


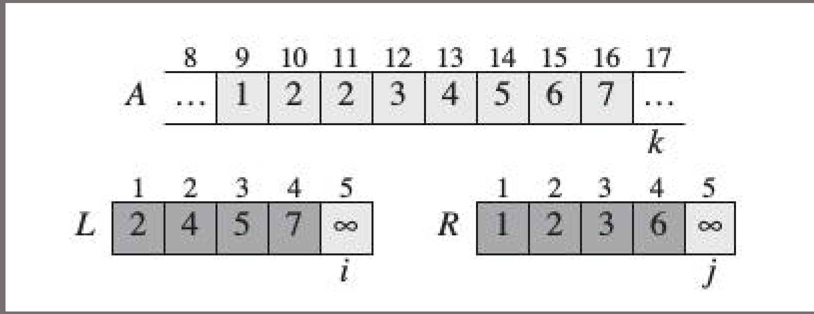
Illustration of Merge in Merge Sort











Merge in Merge Sort

```
MERGE(A, p, q, r)
n1 = q - p + 1
n2 = r - q
let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i - 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```

Proof of Correctness of Merge Operation

Lines 10 – 17, of the algorithm, perform the $r - p + 1$ basic steps by maintaining the following loop invariants:

(i) At the start of each iteration of the for loop of lines 12 – 17, the subarray $A[p..k-1]$ contains $k - p$ smallest elements of $L[1..n_1+1]$ and $R[1..n_2+1]$, in sorted order.

(ii) Moreover, $L[i]$ and $R[j]$ are the smallest elements of their arrays that have not been copied back into A.

To Show

- Loop invariant holds prior to first iteration of for loop of lines 12 – 17
- Each iteration of the loop maintains the invariant
- Invariant provides a useful property to show correctness when the loop terminates

Initialization

- Prior to the first iteration of the loop, we have $k = p$, so that the subarray $A[p..k-1]$ is empty
- This empty subarray contains the $k - p = 0$ smallest elements of L and R, and since $i = j = 1$, both $L[i]$ and $R[j]$ are the smallest elements of their arrays that have not been copied back into A.

Maintenance of Loop Invariant

- Let L[i] ≤ R[j], then L[i] is the smallest element not yet copied back into A
- Because A[p..k−1] contains the k – p smallest elements, after line 14 copies L[i] into A[k], the subarray A[p .. k] will contain the k – p + 1 smallest elements

Termination of Loop Invariant

- At termination, $k = r + 1$. By the loop invariant, the subarray $A[p..k-1]$, which is $A[p..r]$, contains the $k - p = r - p + 1$ smallest elements of $L[1..n_1 +1]$ and $R[1..n_2+1]$, in sorted order
- The arrays L and R together contain $n_1 + n_2 + 2 = r - p + 3$ elements

Termination of Loop Invariant

- All but the two largest have been copied back into A, and these two largest elements are the sentinels.

Time Complexity of Merge Operation

- Each of lines 1 – 3 and 8 – 11 takes constant time
- for loops of lines 4 – 7 takes $\theta(n_1 + n_2) = \theta(n)$ time
- There are n iterations of the for loop of lines 12 – 17, each of which takes constant time
- Here $n = r - p + 1$

Forming Recurrence Relation

- When we have $n > 1$ elements running time is as follows:
- Divide: Just computes the middle of the subarray, which takes constant time. Thus, $D(n) = \theta(1)$
- Conquer: We recursively solve two subproblems, each of size $n/2$, which contributes $2T(n/2)$ to the running time.
- Combine: MERGE procedure on an n–element subarray takes time $\theta(n)$, and so $C(n) = \theta(n)$

Recurrence Relation

- $T(n) = \theta(1)$ if $n = 1$

  $= 2T(n/2) + \theta(n)$ if $n > 1$

Substitution Method

$T(n) = 2T(n/2) + n$ ————————————————————(1)

$T(n/2) = 2 * T(n/2^2) + n/2$ ————————————————————(2)

Substiute (2) in (1)

$T(n) = 2 * (2 * T(n/2^2) + n/2) + n$

$= \quad 2^2 * T(n/2^2) + 2*n$

.....

$T(n) = \quad 2^k * T(n/2^k) + k*n$

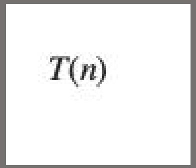
Substitution Method

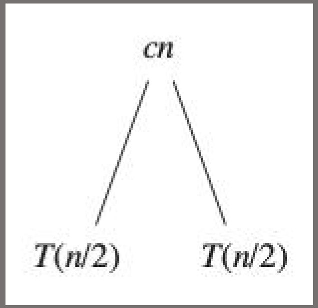
- In the k^{th} iteration, the value of n becomes 1
- therefore $n/2^k = 1$
- $n = 2^k$
- $k = \log n$
- $T(n) = 2^k * T(n/2^k) + k*n$ becomes
- $T(n) = n + n \log n$
- $T(n) = \theta(n \log n) = O(n \log n)$

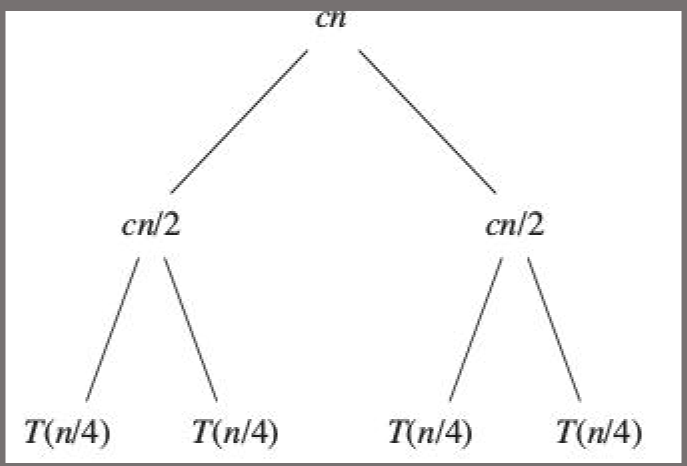
Relationship between Time Functions

$1 < \log(n) < \sqrt{n} < n < n \log n < n^2 < n^3 < \ldots\ldots < 2^n < 3^n \ldots\ldots n! < \ldots\ldots < n^n$

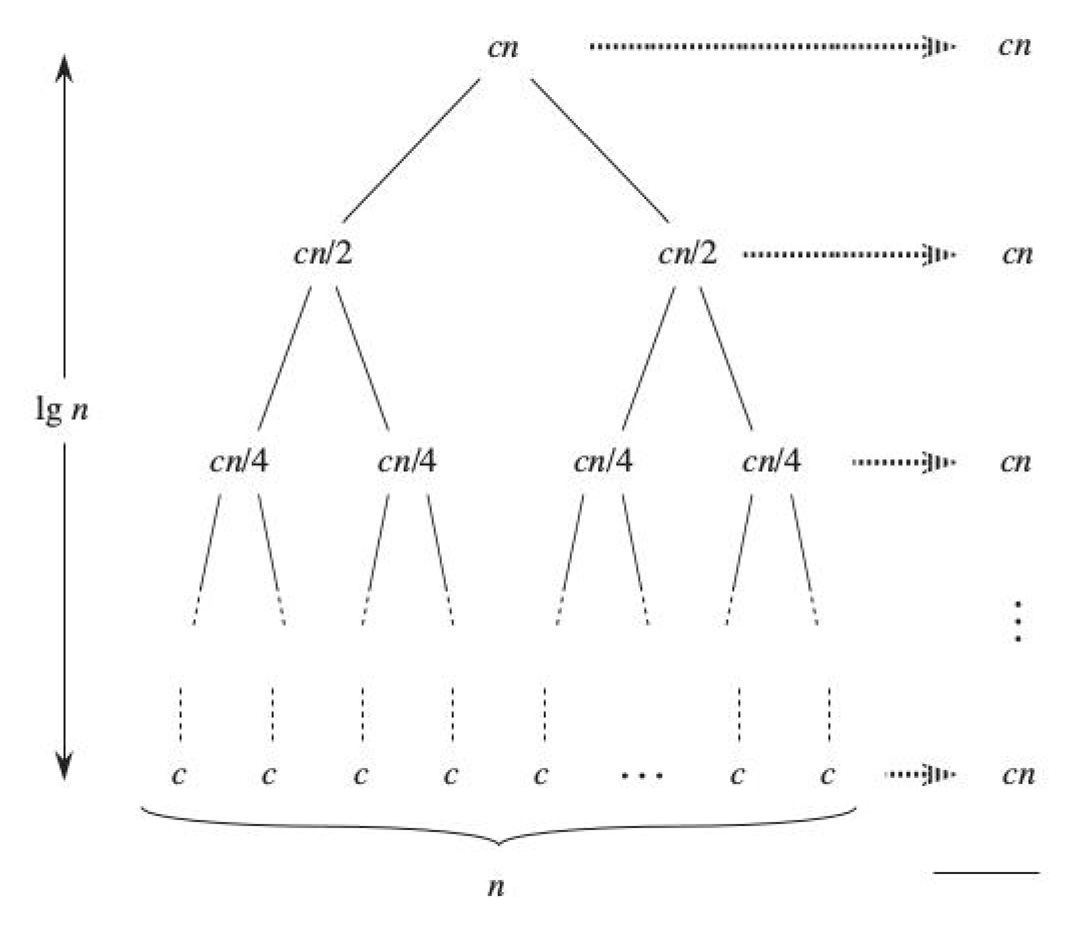
Recurrence Tree







Recurrence Tree



(d)

Master's Theorem for Divide and Conquer

Consider a recurrence relation of the form:

$T(n) = aT(n/b) + \theta(n^k \log^p n)$

1) If $a > b^k$, then $T(n) = \theta(n^{\log_b a})$

2) If $a = b^k$, then

   a. If $p > -1$, then $T(n) = \theta(n^{\log_b a} \log^{p+1} n)$

   b. If $p = -1$, then $T(n) = \theta(n^{\log_b a} \log \log n)$

   c. If $p < -1$ then $T(n) = \theta(n^{\log_b a})$

Master's Theorem for Divide and Conquer

3) If $a<b^k$,

a. If $p \geqslant 0$, then $T(n) = \theta(n^k \log^p n)$

b. If $p < 0$, then $T(n) = \theta(n^k)$