מבני נתונים ואלגוריתמים 2
תירגול: Merge Sort

Based on: Introduction to Algorithms
by Cormen, Leiserson, Rivest, and Stein

Growth of Functions – O, o, Ω, ω & Θ notations

- $f(n) \in O\big(g(n)\big)$ **if there exist constants** c and n_0 such that $f(n) \le c \cdot g(n)$ for all $n \ge n_0$
  - Growth of $f(n)$ is not faster than growth of $g(n)$

- $f(n) \in \Omega\big(g(n)\big)$ **if there exist constants** c and n_0 such that $f(n) \ge c \cdot g(n)$ for all $n \ge n_0$
  - Growth of $f(n)$ is not slower than growth of $g(n)$

- $f(n) \in \Theta\big(g(n)\big)$ if and only if $f(n) \in O\big(g(n)\big)$ and $f(n) \in \Omega\big(g(n)\big)$
  - Growth of $f(n)$ is the same as growth of $g(n)$

Growth of Functions – O, o, Ω, ω & θ notations

- $f(n) \in o(g(n))$ **if for any constant** c, there exists a constant n_0 such that $f(n) < c \cdot g(n)$ for all $n \geq n_0$
  - Growth of $f(n)$ is strictly smaller than growth of $g(n)$

- $f(n) \in \omega(g(n))$ **if for any constant** c, there exists a constant n_0 such that $f(n) > c \cdot g(n)$ for all $n \geq n_0$
  - Growth of $f(n)$ is strictly faster than growth of $g(n)$

Common abuse of notation:

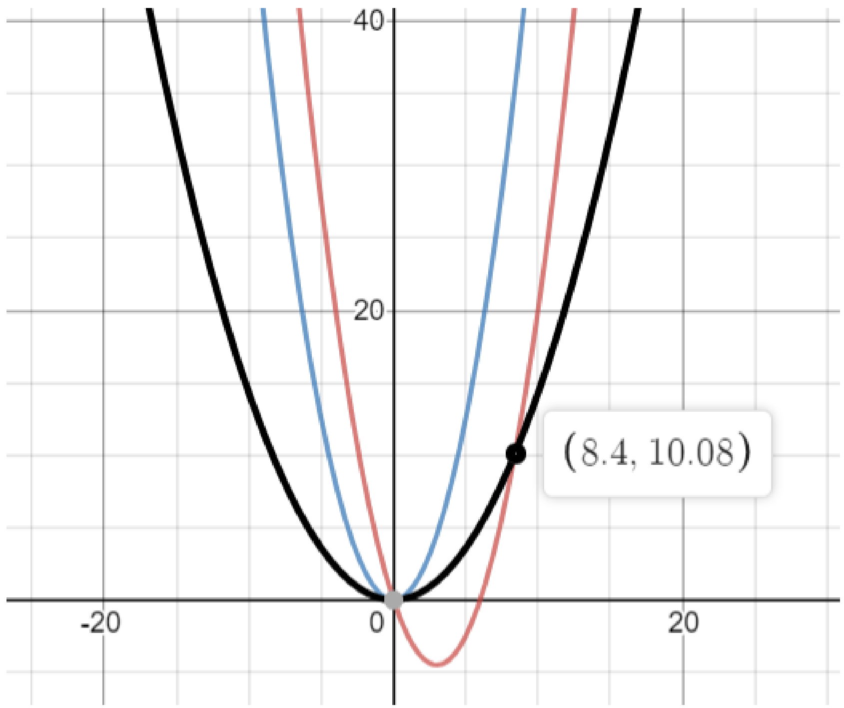
- We use $f(n) = O(g(n))$ instead of $f(n) \in O(g(n))$
- In the algorithmic literature people write $f(n) = \Omega(g(n))$ instead of writing $f(n) \notin o(g(n))$

Growth of Functions - O,o,Ω, ω & θ notations

Example:

- $0.5n^2 - 3n = \Theta(n^2)$ for c_1=1/7, c_2=1/2, $n_0 = 8.4$

- $\mathbf{0.5n^2 - 3n}$
- **(1/2)*** $\boldsymbol{n^2}$
- **(1/7)*** $\boldsymbol{n^2}$



Example

- Question: what can we say about the following function?

$$f(n) = \begin{cases} n^2, & n \text{ is even} \\ 1, & n \text{ is odd} \end{cases}$$

- $f(n) \in O(n^2)$ since $f(n) \le n^2$, for every n.
- $f(n) \notin O(1)$ since for any constant c there are infinitely many n's for which $f(n) > c \cdot 1$.

- $f(n) \in \Omega(1)$ since $f(n) \ge 1$, for every n.
- $f(n) \notin \Omega(n^2)$ since for any constant c there are infinitely many n's for which $f(n) < c \cdot n^2$.

- $f(n) \notin o(n^2)$ since for $c = 1$ there are infinitely many n's for which $f(n) \ge c \cdot n^2$.

Divide and conquer approach

- Many algorithms are *recursive* in structure

- They typically follow a *divide-and-conquer* approach:

1. **Divide** the problem into smaller sub problems.

2. **Conquer** the sub problems by solving them recursively.

3. **Combine** the solutions to the sub problems into the solution for the original problem.

Algorithm Merge Sort

- Merge Sort is an efficient sort algorithm which follows the divide-and-conquer paradigm:

1. **Divide:** Divide the n-element sequence to be sorted into two subsequences of (n/2) elements.

2. **Conquer:** Sort the two subsequences recursively using merge sort.

3. **Combine:** Merge the two sorted subsequences to produce the sorted answer.

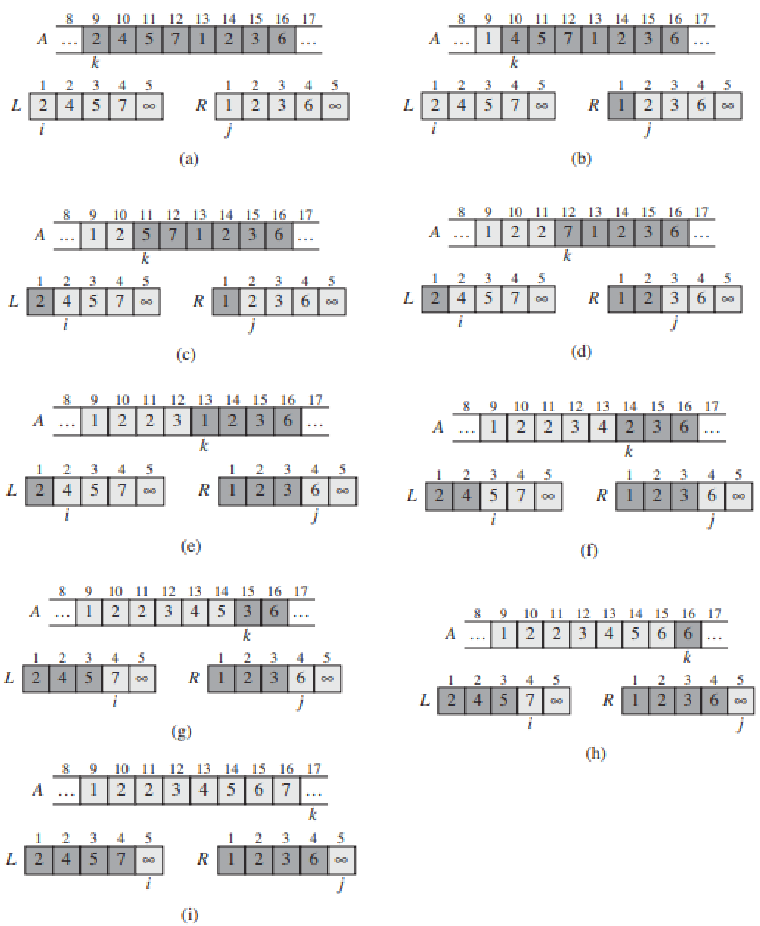
Algorithm Merge Sort-pseudocode

```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

- A is n-element array to be sorted
- p, q and r – indices into the array, $p \le q < r$
- Initial call to sort A: Merge-Sort(A,1,n)
- The recursion terminates when p=r ==> length(sequence to be sorted)=1

Algorithm Merge -pseudocode

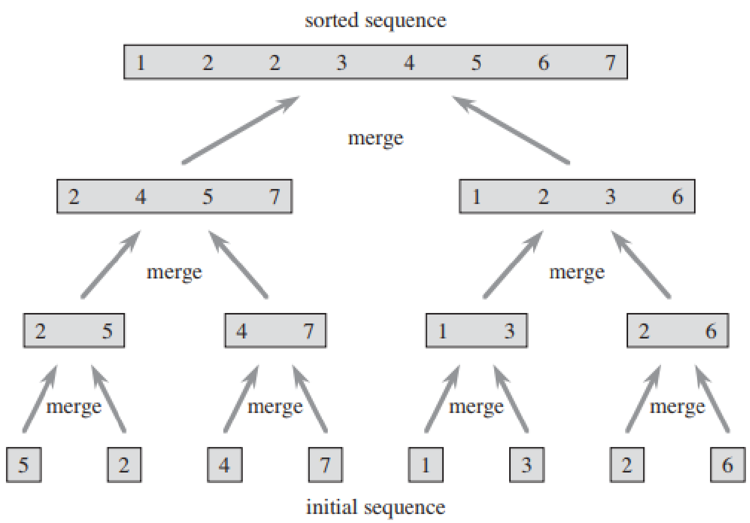
- Merge is the "combine" step and key operation of the merge sort
- Merge procedure assumes that A[p..q] and A[q+1…r] are in sorted order

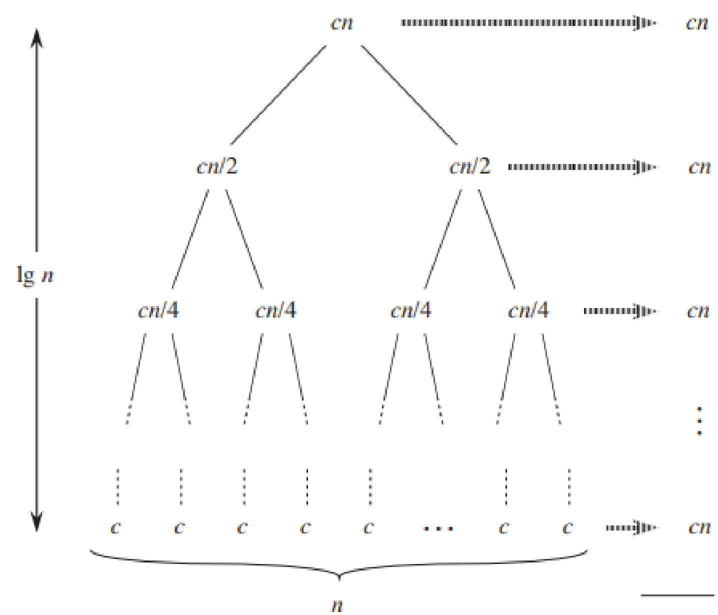


MERGE(A, p, q, r)

```
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
```

Algorithm Merge Sort - example





Algorithm Merge - running time

```
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
```

c1

Θ(n1+n2)=Θ(n)

c2

Θ(n1+n2)=Θ(n)

Θ(n)

Algorithm Merge Sort - running time

- Denote: T(n)- total running time of Merge Sort

MERGE-SORT(A, p, r)

```
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```

Divide: D(n)= Θ(1)

Conquer: 2T(n/2)

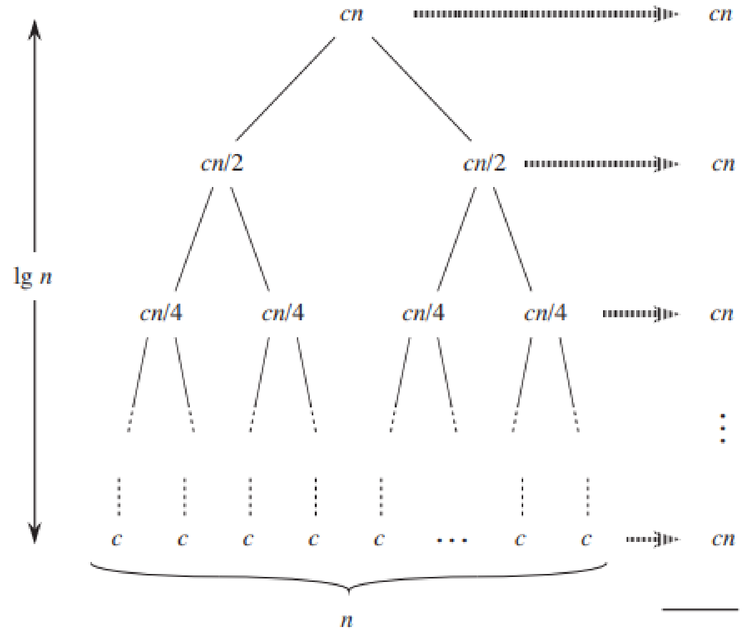
Merge: C(n)=Θ(n)

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 , \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 . \end{cases} \longrightarrow T(n) = \begin{cases} c & \text{if } n = 1 , \\ 2T(n/2) + cn & \text{if } n > 1 , \end{cases}$$

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) = 2 * \left(2T\left(\frac{n}{4}\right) + \Theta\left(\frac{n}{2}\right)\right) + \Theta(n) = \cdots =$$

$$= \sum_{i=0}^{logn} 2^i \Theta\left(\frac{n}{2^i}\right) = \Theta(n) \sum_{i=0}^{logn} 1 = \Theta(n)(logn + 1) = \Theta(n\ log\ n)$$

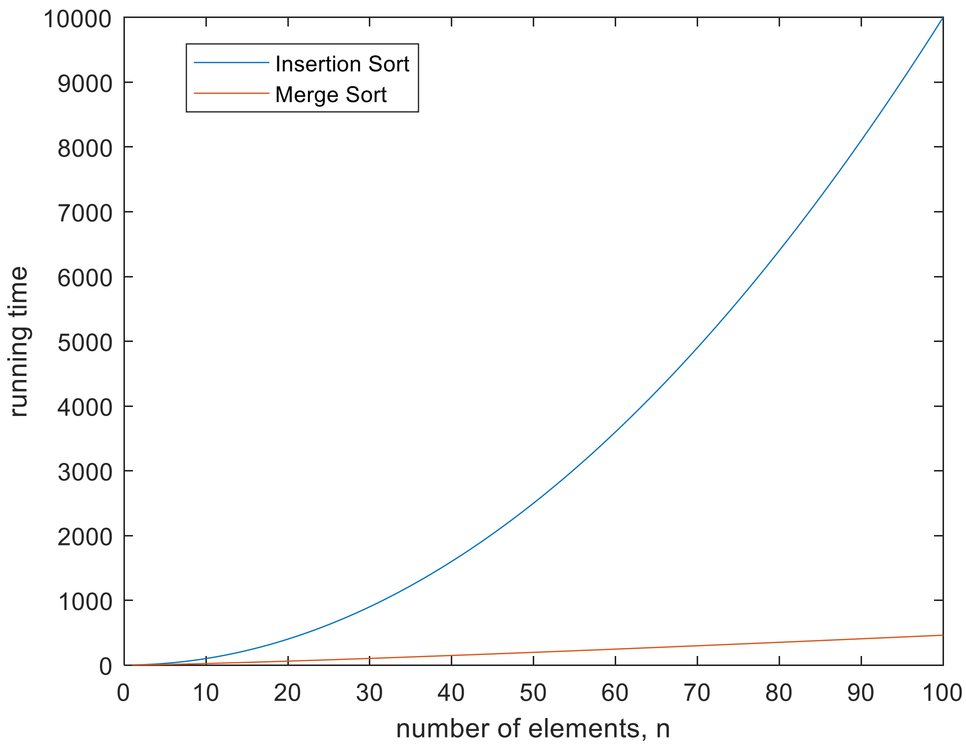
Algorithm Merge Sort - recursion tree



- The total number of levels in the recursion tree: log n + 1 ,
  n = number of leaves = input size
- Running time at each level : at level i we have 2^i nodes, while each contributes a cost of $cn/2^i$, so total cost = $2^i cn/2^i$ = cn
- Total running time: cn(log n +1)= cn log n + cn = Θ(n log n)

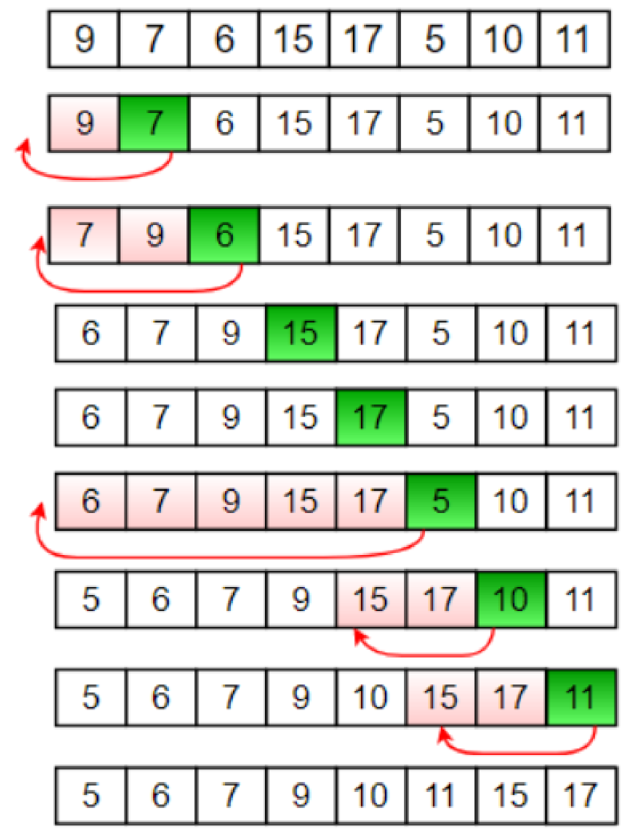
Sort algorithms - comparison

- Merge Sort: $\Theta(n \log n)$
- Insertion Sort: $\Theta(n^2)$



Algorithm Insertion Sort-reminder

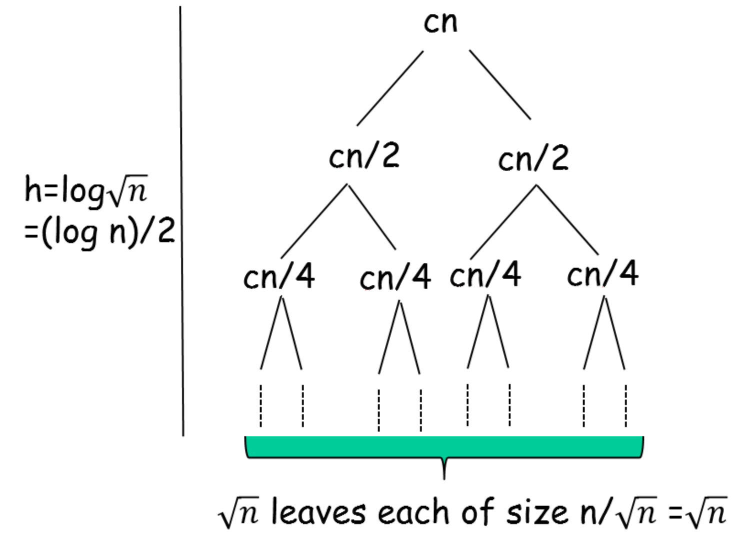
```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 .. j − 1].
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```



Example 1

- Given the algorithm which combines Merge Sort with Insertion Sort:

```
MS-IS(A,p,r)
{
    if (r − p + 1 ≤ √size(A) )
       then  Insertion-Sort(A,p,r)
    else
    {
        q ← ⌊(r + p)/2⌋
        MS-IS(A,p,q)
        MS-IS(A,q+1,r)
        Merge(A,p,q,r)
    }
}
```



- Show that running time of MS-IS is $\Theta(n^{3/2})$.

$$T(n) = \sum_{i=0}^{(log n)/2} \Theta(n) + \sqrt{n}\Theta((\sqrt{n})^2) = \Theta(nlog\, n) + \Theta(n^{3/2}) = \Theta(n^{3/2})$$

Merge IS

Merge Sort Demo

- https://www.youtube.com/watch?v=XaqR3G_NVoo