5118006-03 Data Structures

Sorting

31 May 2024

Shin Hong

Motivation: Search

- Checking whether an item exists in a list of *n* items, or not

- Sequential search
  - check each item by iterating over the list

- Binary search
  - assume that the list is sorted
  - compare the middle point of the list with the target item, and continue the binary search on a half of the list where the item may be found

Binary Search Algorithm

Input:

a given sorted list of *N* elements, *L*

a target value, *T*

Output:

the index of T, or *undefined*

Algorithm:

```
left = 0, right = N – 1
while left <= right do
    mid = (left + right) / 2
    if L[mid] == T
        return mid
    else if L[mid] < T
        right = mid – 1
    else /* if T < L[mid] */
        left = mid + 1
end while
return undefined
```

Sorting Problem

- Given a list of n items $(a_1, a_2, \cdots, a_n)$, find a permutation σ $(a_{\sigma 1}, a_{\sigma 2}, ..., a_{\sigma n})$ such that $key(a_{\sigma i}) \leq key(a_{\sigma\, i+1})$ for $1 \leq i < n$
  - assume that the equivalence and the ordering in items are well defined
- A sorting is stable if the permutation satisfies the following condition:

  if $key(a_{\sigma i}) = key(a_{\sigma j})$ and $i < j$, $\sigma_i < \sigma_j$

Insertion Sort

- Given an unsorted list *U* of items, repeat the following two steps until the list becomes empty:
  (1) remove a minimum/maximum item in the *U*
  (2) insert the removed item to the sorted list

- We can use a prefix *U*[0 .. *i*] as a sorted list for *i*-th turn to make sorting happen within the given list

Algorithm

- Algorithm

  Input: a given a list of N items, $L[0 .. N\text{-}1]$

  Output: a sorted list of item L

  Procedure

  for i in $[0 .. N\text{-}2]$

  find j such that L[j] is minimum among $L[i .. N\text{-}1]$

  swap $L[i]$ and $L[j]$

  end for

- Example

(7, b)	(3, d)	(2, e)	(9, a)	(5, g)	(3, a)	(2, h)

Analysis

- Algorithm

  Input: a given a list of N items, $L[0 .. N\text{-}1]$

  Output: a sorted list of item L

  Procedure

  for i in $[0 .. N\text{-}2]$

  find j such that L[j] is minimum among $L[i .. N\text{-}1]$

  swap $L[i]$ and $L[j]$

  end for

- Time complexity

$$O\left(\sum_{i=0}^{N-2}(i+1)\right) = O(n^2)$$

Merge Sort

- To sort a sequence S with n elements:

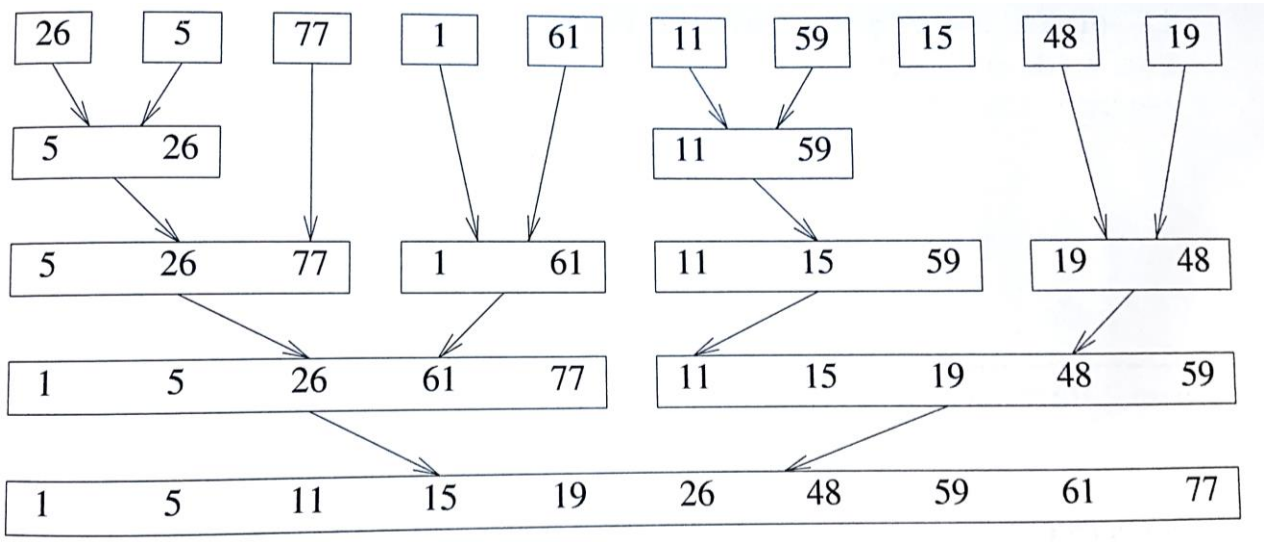
  1. Return S immediately if n is zero or one.

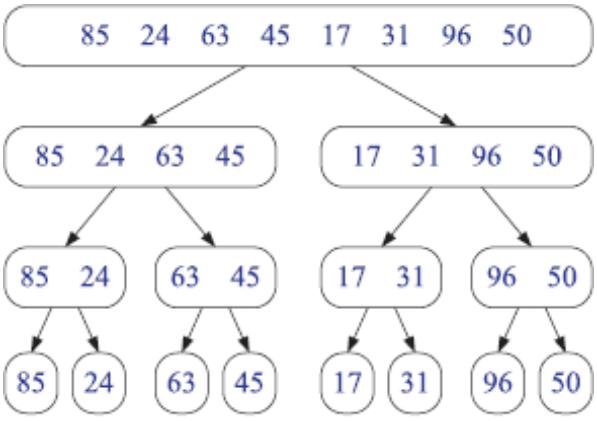
  Otherwise, create S_1 and S_2 by dividing S evenly

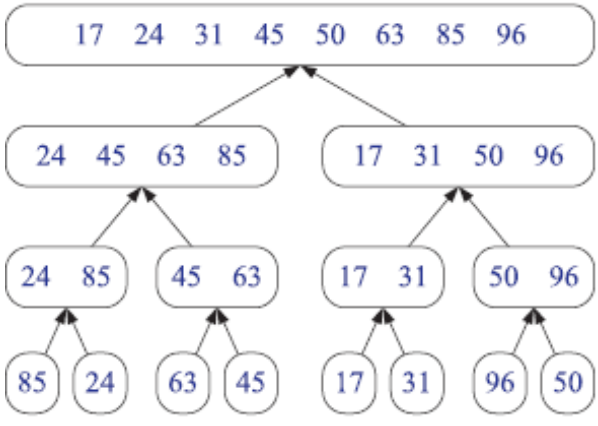
  2. Sort S_1 and S_2, recursively

  3. Put back the elements into S by merging S_1 and S_2

Examples


26 5 77 1 61 11 59 15 48 19
5 26 11 59
5 26 77 1 61 11 15 59 19 48
1 5 26 61 77 11 15 19 48 59
1 5 11 15 19 26 48 59 61 77


85 24 63 45 17 31 96 50
85 24 63 45 17 31 96 50
85 24 63 45 17 31 96 50
85 24 63 45 17 31 96 50


17 24 31 45 50 63 85 96
24 45 63 85 17 31 50 96
24 85 45 63 17 31 50 96
85 24 63 45 17 31 96 50

Quick Sort

- Given a list of items $L[0 .. N-1]$, reorder L such that all elements in $L[0 .. p]$ are less than equal to all elements in $L[p+1 .. N-1]$, and sort two sublists $L[0.. p]$ and $L[p+1 .. N-1]$ independently

- Example

5	3	1	9	7	3	2	2

Quick Sort: Algorithm

Sort ($L[0 .. N\text{-}1]$, *left*, *right*)

if left == right **then return**

$p = left$; $l = left + 1$; $r = right$

while $l < r$ **begin**

while $L[l] < L[p] \wedge l \leq right$ **begin** l++ **end while**

while $L[p] < L[r] \wedge left + 1 \leq r$ **begin** r-- **end while**

if $l < r$ **then** swap $L[l]$ and $L[r]$; l++ ; r--

end while

swap $L[p]$ and $L[r]$

Sort (L, *left*, $r - 1$)

Sort (L, $r + 1$, *right*)