CSCI-SHU 210 Data Structures

Recitation13 Worksheet Sorting Algorithms

Sorting related algorithms

1. Merge-Sort

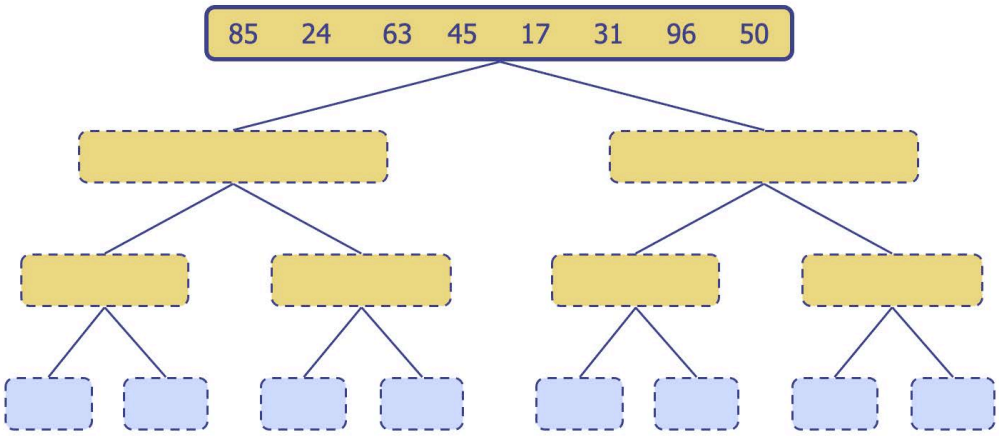
`Understand Python's array-based implementation of merge-sort algorithm.`

```
1  def merge(S1, S2, S):
2    """Merge two sorted Python lists S1 and S2 into properly sized list S."""
3    i = j = 0
4    while i + j < len(S):
5      if j == len(S2) or (i < len(S1) and S1[i] < S2[j]):
6        S[i+j] = S1[i]                # copy ith element of S1 as next item of S
7        i += 1
8      else:
9        S[i+j] = S2[j]                # copy jth element of S2 as next item of S
10       j += 1
```

```
1  def merge_sort(S):
2    """Sort the elements of Python list S using the merge-sort algorithm."""
3    n = len(S)
4    if n < 2:
5      return                          # list is already sorted
6    # divide
7    mid = n // 2
8    S1 = S[0:mid]                     # copy of first half
9    S2 = S[mid:n]                     # copy of second half
10   # conquer (with recursion)
11   merge_sort(S1)                    # sort copy of first half
12   merge_sort(S2)                    # sort copy of second half
13   # merge results
14   merge(S1, S2, S)                  # merge sorted halves back into S
```

Suppose you are given an input sequence [85, 24, 63, 45, 17, 31, 96, 50]. Visualize the merge-sort algorithm using the mert-sort tree.

Important:
There are two parts, one is the recursion part which does the partition of the input sequence into 2, 4, 8...until reaching the base case; another one is the merging part, which merges sorted halves back into S.



2. In-place Quick-Sort

Task1. Understand in-place quick-sort algorithm.

```
1  def inplace_quick_sort(S, a, b):
2    """Sort the list from S[a] to S[b] inclusive using the quick-sort algorithm."""
3    if a >= b: return                          # range is trivially sorted
4    pivot = S[b]                               # last element of range is pivot
5    left = a                                   # will scan rightward
6    right = b-1                                # will scan leftward
7    while left <= right:
8      # scan until reaching value equal or larger than pivot (or right marker)
9      while left <= right and S[left] < pivot:
10       left += 1
11     # scan until reaching value equal or smaller than pivot (or left marker)
12     while left <= right and pivot < S[right]:
13       right -= 1
14     if left <= right:                          # scans did not strictly cross
15       S[left], S[right] = S[right], S[left]                  # swap values
16       left, right = left + 1, right - 1                      # shrink range
17
18   # put pivot into its final place (currently marked by left index)
19   S[left], S[b] = S[b], S[left]
20   # make recursive calls
21   inplace_quick_sort(S, a, left - 1)
22   inplace_quick_sort(S, left + 1, b)
```

Task2. Given an input sequence [11, 52, 30, 25, 4, 18, 63, 20], give the intermediate output after executing line 1 up to line 19 (without running the recursive calls part).