CE2101/ CZ2101: Algorithm Design and Analysis

Week 2: Review Lecture

Ke Yiping, Kelly

Content

- Mergesort
- Quicksort

Mergesort

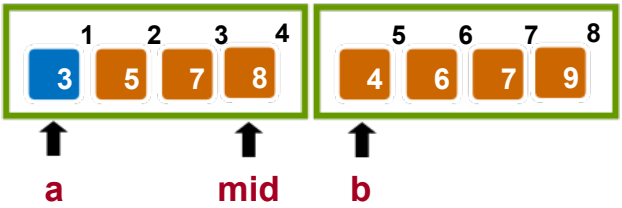
- Uses the Divide and Conquer approach.
- It recursively divide a list into two halves of approximately equal sizes, until the sub-list is too small (no more than two elements).
- Then, it recursively merges two sorted sub-lists into one sorted list.

```
void mergesort(int n, int m)
{   int mid = (n+m)/2;
  if (m-n <= 0)
     return;
  else if (m-n > 1) {
     mergesort(n, mid);
     mergesort(mid+1, m);
  }
  merge(n, m);
}
```

Mergesort – Merge Function

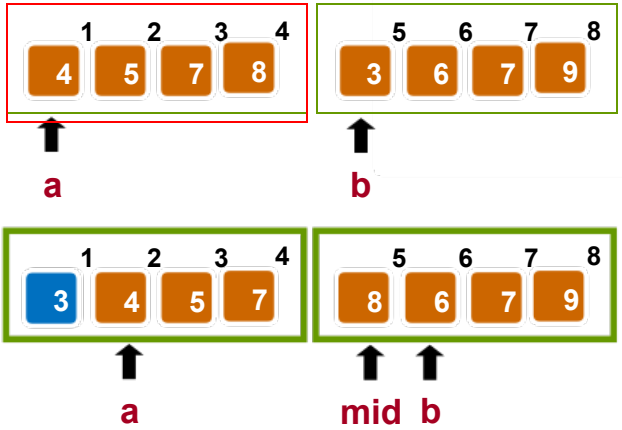
- In-place version: ***merging*** is performed directly on the original array; swapping and shifting are needed

Case 1: if slot[a] < slot[b], there is nothing much to do since smaller element already in correct position (with regard to the merged array)



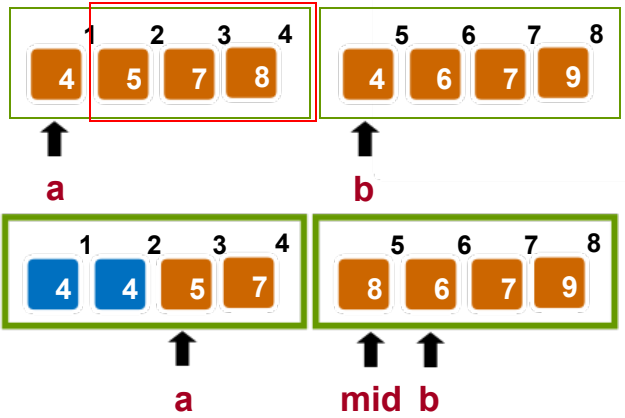
Mergesort – Merge Function

Case 2: if slot[a] > slot[b], then Right-shift (by one) elements of left subarray from index a to 'mid' and insert element at slot[b] into slot[a]



Mergesort – Merge Function

Case 3: if slot[a] == slot[b], then slot[a] is in the correct position. So, move slot[b] next to beside slot[a], by Right-shifting and swapping



Mergesort - Complexity

- The worst-case running time for merging two sorted lists of total size n is $n - 1$ key comparisons.
- The running time of Mergesort is $O(n \lg n)$.
  - Recurrence equation
  - Recursion tree

Evaluation of Mergesort

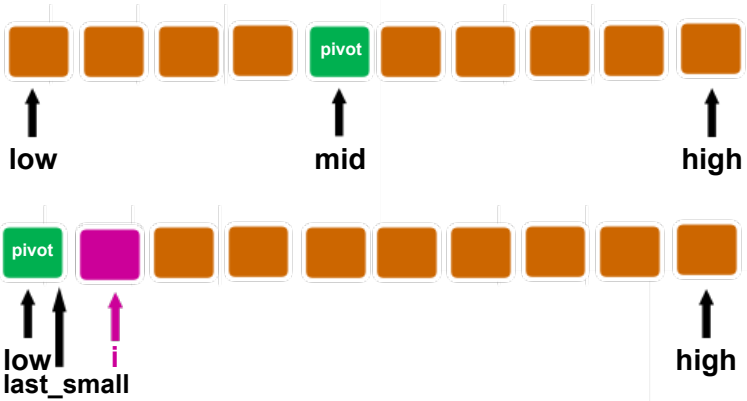
- **Strengths:**
  - Simple and good runtime behavior
  - Easy to implement when using linked list
- **Weaknesses:**
  - Difficult to implement for contiguous data storage such as array without auxiliary storage (requires data movements during merging)

Quicksort

- Uses the “Divide and Conquer” approach
- Partition function splits an input list into two sub-lists by comparing all elements with the pivot:
  - Elements on the left < pivot
  - Elements on the right ≥ pivot

```
void quicksort(int n, int m)
{
   int pivot_pos;
   if (n >= m)
      return;
   pivot_pos = partition(n, m);
   quicksort(n, pivot_pos - 1);
   quicksort(pivot_pos + 1, m);
}
```

Quicksort – Partition Function

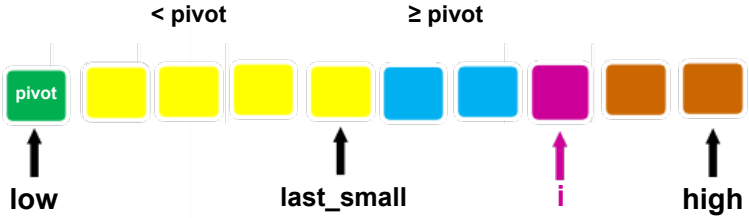


First step:
swap(low, mid);

Quicksort – Partition Function

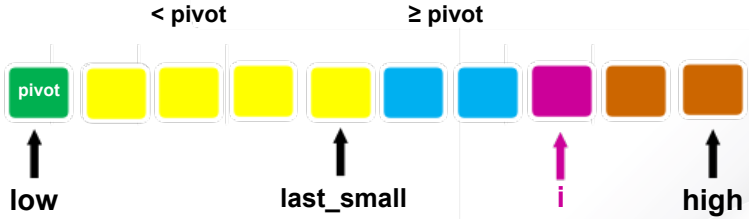
- **Case 1: Current element < pivot**

  swap(++last_small, i);

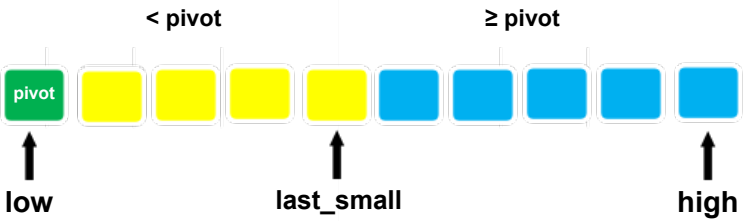


- **Case 2: Current element ≥ pivot**

  i++;



Quicksort – Partition Function



Last step: swap(low, last_small);

Quicksort - Complexity

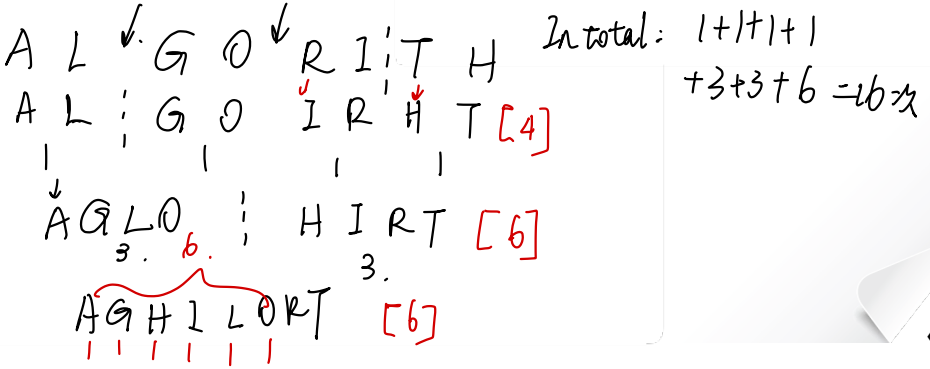
- The worst-case time complexity of Quicksort is □(*n*2).
- The best-case and average-case time complexities of Quicksort are both □(*n*lg*n*).

Quicksort's Performance

- Strengths:
  - Fast on average
  - No merging required
  - Best case occurs when pivot always splits array into equal halves
- Weaknesses:
  - Poor performance when pivot does not split the array evenly
  - Quicksort also performs badly when the size of list to be sorted is small
  - If more work is done to select pivot carefully, the bad effects can be reduced

Exercise

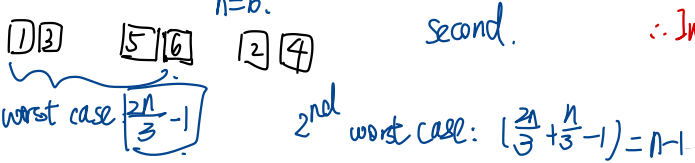
- Manually execute Mergesort to sort the list of characters (A, L, G, O, R, I, T, H) in alphabetical order. How many comparisons between the characters will be performed? Explain your answer by illustrating the changes of the list content during the execution of Mergesort. **[AY1617S2]**



Exercise

- Consider the following modification to MergeSort, called **3-way MergeSort**. It divides the array into three subarrays (as equally-sized as possible), sorts them recursively, and then calls a **3-way merge** function. The 3-way merge function combines three sorted subarrays into one sorted array in the following way. It merges the first two subarrays by calling the merge function in the original MergeSort. It then merges the resultant array with the third subarray, again by calling the original merge function.

- Given three equally-sized sorted subarrays with *n* elements in total as input, what is the worst-case number of key comparisons in the 3-way merge function? Briefly justify your answer. **[AY1819S1]**



Exercise

- Suppose Quicksort always picks the left-most element of an array as the pivot. If the input array is already sorted in ascending order, what is the time complexity of Quicksort? Justify your answer. **[AY1617S1]**

ith element: $(n-i)$ comparisons.

$$\sum 1 + \ldots + (n-1) = \frac{n(n-1)}{2} = \frac{n^2-n}{2}$$

Exercise

Consider an input array that contains five integers {1, 2, 3, 4, 5}. Give an initial arrangement of the keys in the array such that when it is input to Quicksort, the subarray with keys smaller than the pivot at each call of the partition function is always empty. You should use the Quicksort algorithm learnt in the lectures. **[AY1819S1]**

worst case scenario.