Principles of Computer Science

Sorting

Today's Topics

- Merge Sort
  - The Efficiency of Merge Sort
- Quick Sort
  - The Efficiency of Quick Sort

Sub-List

- Given a list l, define a sublist l[i:j], which contains elements i, i+1, i+2, …, j-2, j-1, j of the original list.
- Examples
  - `L = [a b c d e f g h i j k l m n o]`
  - `L[0:3] = [a b c d]`
  - `L[3:6] = [d e f g]`

Merge Sort

- Divide an array into halves
  - Sort the two halves
  - Merge them into one sorted array
- An example of the divide and conquer algorithm
  - This is often part of a recursive algorithm
  - However recursion is not a requirement

Merge Two Sorted List

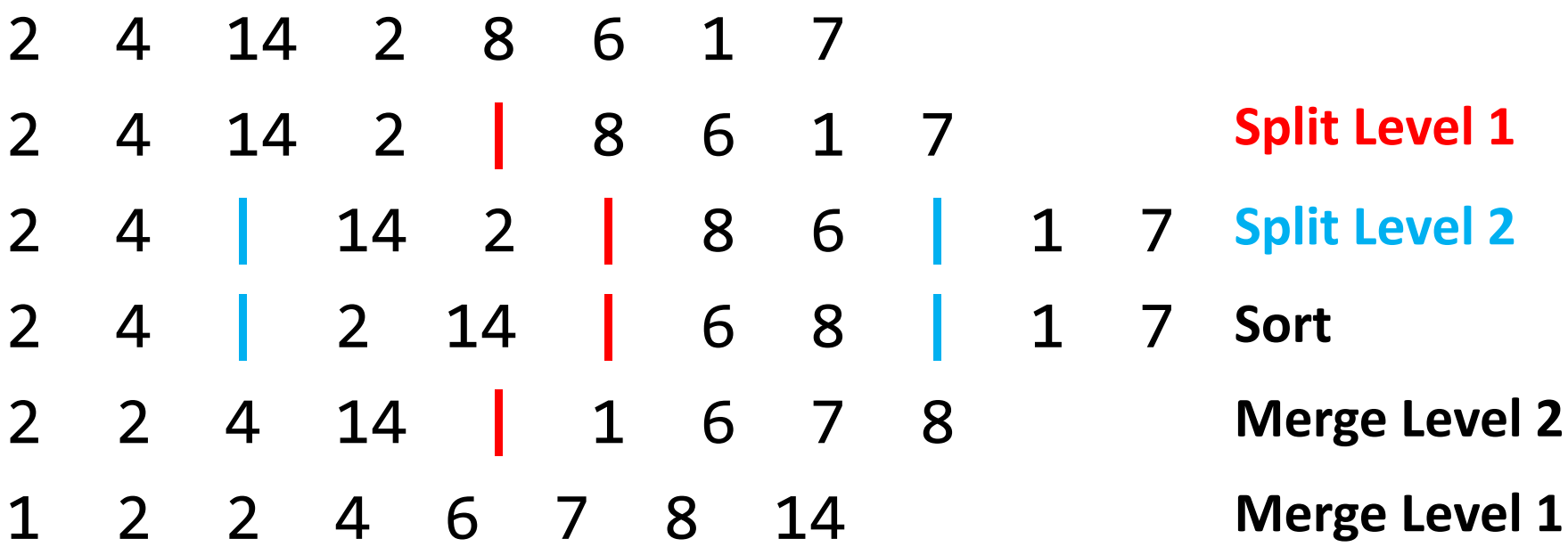
```
List merge(List sortedList1, List sortedList2)
{
  // Assume the sortedList1 and sortedList2
  // are properly sorted.

  // Merge them into another sorted list.

  // Return the merged sorted list
}
```

Merge Sort: Recursive Formulation

- Define a recursive algorithm
  - Uses merge() to merge to sorted lists
  - Uses sub-lists

```
List mergeSort(List input) {
  N = length of input;
  if( N == 1 ) return input;
  else return
    merge(
      mergeSort(input[0:(N/2)]),
      mergeSort(input[(N/2+1):N])
    );
}
```

Merge Sort


2 4 14 2 8 6 1 7
2 4 14 2 | 8 6 1 7
Split Level 1
2 4 | 14 2 | 8 6 | 1 7
Split Level 2
2 4 | 2 14 | 6 8 | 1 7
Sort
2 2 4 14 | 1 6 7 8
Merge Level 2
1 2 2 4 6 7 8 14
Merge Level 1

What is the runtime of Merge Sort?

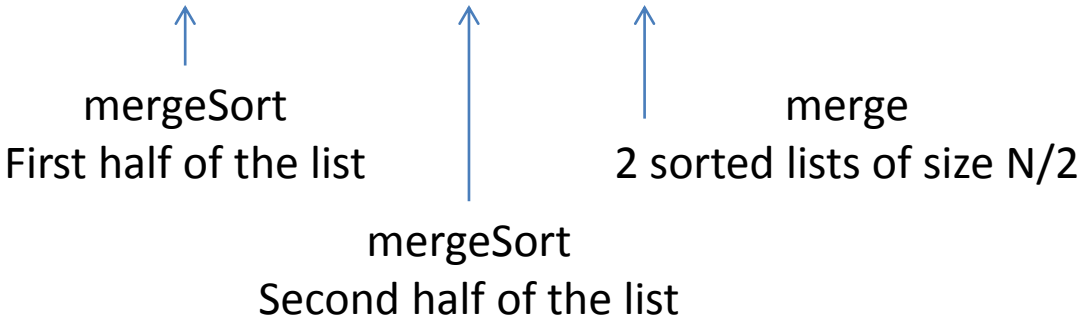
- Let the input list be of length N.
- For simplicity, let's say N is even.
  - Similar reasoning when N is odd.

What is the runtime of Merge Sort?

- Let the input list be of length N.
- For simplicity, let's say N is even.
  - Similar reasoning when N is odd.
- If N = 1, then T(1) = 1.

What is the runtime of Merge Sort?

- Let the input list be of length N.
- For simplicity, let's say N is even.
  - Similar reasoning when N is odd.
- If N = 1, then T(1) = 1.
- If N > 1, and N is even, then
  - $T(N) = T(N/2) + T(N/2) + N$



Need to Solve Recursive Equation

$$T(N) = 2\,T(N/2) + N$$

Need to Solve Recursive Equation

$$T(N)$$
$$= 2\,T(N/2) + N$$
$$= 2\,(2\,T(N/4) + N/2) + N = 4\,T(N/4) + 2\,N$$

Need to Solve Recursive Equation

$$
\begin{aligned}
&T(N) \\
&= 2\,T(N/2) + N \\
&= 2\,(2\,T(N/4) + N/2) + N = 4\,T(N/4) + 2\,N \\
&= 4\,(2\,T(N/8) + N/4) + 2N = 8\,T(N/8) + 3N \\
&= \ldots
\end{aligned}
$$

Need to Solve Recursive Equation

$$
\begin{aligned}
& T(N) \\
&= 2\,T(N/2) + N \\
&= 2\,(2\,T(N/4) + N/2) + N = 4\,T(N/4) + 2\,N \\
&= 4\,(2\,T(N/8) + N/4) + 2N = 8\,T(N/8) + 3N \\
&= \ldots \\
&= 2^i\,T(N/2^i) + iN
\end{aligned}
$$

Need to Solve Recursive Equation

$$
\begin{aligned}
&T(N) \\
&= 2\,T(N/2) + N \\
&= 2\,(2\,T(N/4) + N/2) + N = 4\,T(N/4) + 2\,N \\
&= 4\,(2\,T(N/8) + N/4) + 2N = 8\,T(N/8) + 3N \\
&= \ldots \\
&= 2^i\,T(N/2^i) + iN
\end{aligned}
$$

But, $N/2^i > 0$. So the process terminates when $N/2^i = 1$.

Need to Solve Recursive Equation

$$
\begin{aligned}
&T(N) \\
&= 2\,T(N/2) + N \\
&= 2\,(2\,T(N/4) + N/2) + N = 4\,T(N/4) + 2\,N \\
&= 4\,(2\,T(N/8) + N/4) + 2N = 8\,T(N/8) + 3N \\
&= \ldots \\
&= 2^i\,T(N/2^i) + iN
\end{aligned}
$$

But, $N/2^i > 0$. So the process terminates when $N/2^i = 1$.

➔ $2^i = N$

Need to Solve Recursive Equation

$$
\begin{aligned}
&T(N) \\
&= 2\,T(N/2) + N \\
&= 2\,(2\,T(N/4) + N/2) + N = 4\,T(N/4) + 2\,N \\
&= 4\,(2\,T(N/8) + N/4) + 2N = 8\,T(N/8) + 3N \\
&= \ldots \\
&= 2^i\,T(N/2^i) + iN
\end{aligned}
$$

But, $N/2^i > 0$. So the process terminates when $N/2^i = 1$.

➔ $2^i = N$

➔ $i = \log(N)$, eventually

Need to Solve Recursive Equation

$T(N)$
$= 2\,T(N/2) + N$
$= 2\,(2\,T(N/4) + N/2) + N = 4\,T(N/4) + 2\,N$
$= 4\,(2\,T(N/8) + N/4) + 2N = 8\,T(N/8) + 3N$
$= \ldots$
$= 2^i\,T(N/2^i) + iN$

But, $N/2^i > 0$. So the process terminates when $N/2^i = 1$.

➔ $2^i = N$

➔ $i = \log(N)$, eventually

➔ $T(N) = 2^{\log(N)}\,T(1) + \log(N)\,N$
$= N + N\log(N)$

Need to Solve Recursive Equation

$T(N)$
$= 2\,T(N/2) + N$
$= 2\,(2\,T(N/4) + N/2) + N = 4\,T(N/4) + 2\,N$
$= 4\,(2\,T(N/8) + N/4) + 2N = 8\,T(N/8) + 3N$
$= \ldots$
$= 2^i\,T(N/2^i) + iN$

But, $N/2^i > 0$. So the process terminates when $N/2^i = 1$.

➔ $2^i = N$

➔ $i = \log(N)$, eventually

➔ $T(N) = 2^{\log(N)}\,T(1) + \log(N)\,N$
$= N + N\log(N)$

➔ $T(N)$ is $O(N\log(N)$ *Prove it.*

Merge Sort Efficiency

- Merge sort is O(*n* log *n*) in all cases
- It's need for a temporary array is a disadvantage

Compare to Bubble Sort

- Merge Sort is fundamentally better than bubble sort
- Why?
  - N logN is better than N^2

Quick Sort (a naïve version)

- Another recursive sorting algorithm
- Define two helper functions similar to sub-list:
  - Given a list of elements: **list**
  - An element in the list: **pivot**
  - smallerHalf(**list, pivot**): the elements smaller than or equal to pivot
  - biggerHalf(**list, pivot**): the elements bigger than pivot

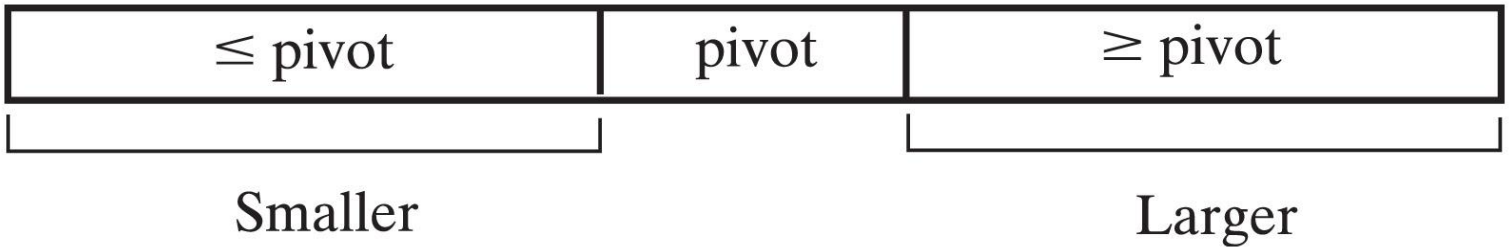
Quick Sort: Steps

- Pick an element, called a pivot, from the list.
- Reorder the list so that all elements which are less than the pivot come before the pivot and so that all elements greater than the pivot come after it (equal values can go either way).
- Recursively sort the sub-list of lesser elements and the sub-list of greater elements.
  - The base case of the recursion are lists of size zero or one.

Quick Sort

- Divides the array into two pieces
  - Not necessarily halves of the array
  - An element of the array is selected as the pivot
- Elements are rearranged so that:
  - The pivot is in its final position in sorted array
  - Elements in positions before pivot are less than the pivot
  - Elements after the pivot are greater than the pivot

Quick Sort

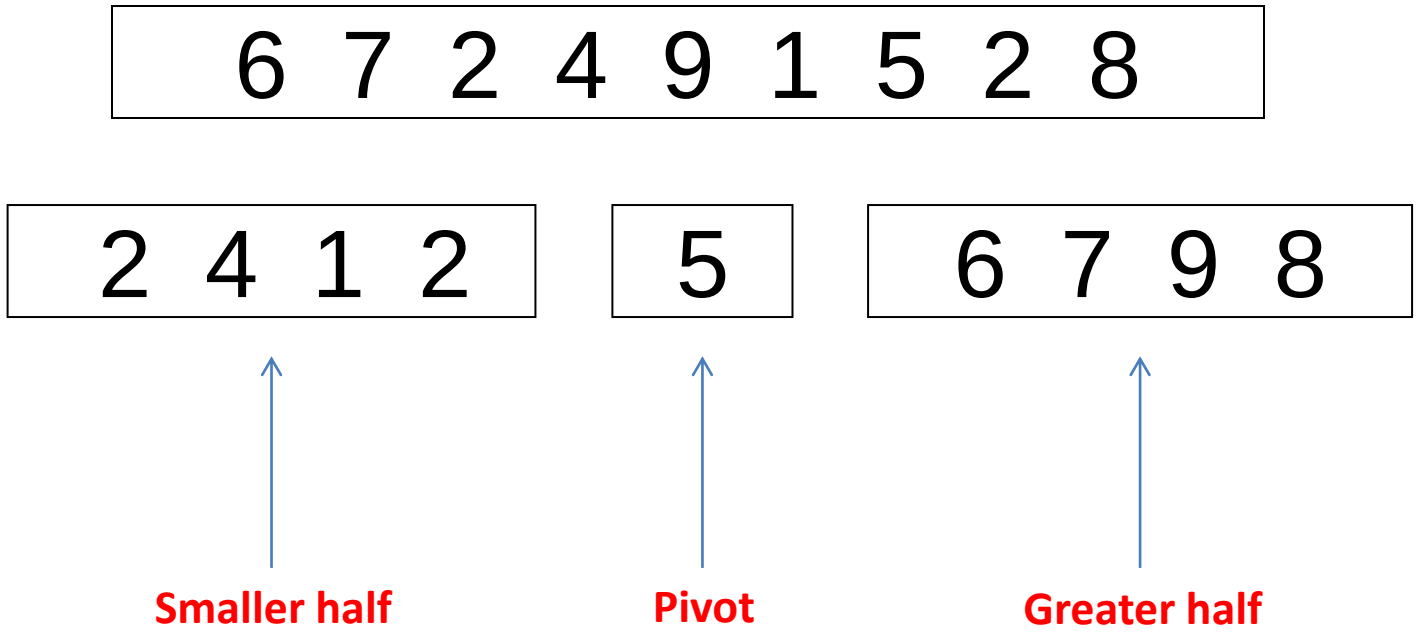


A partition of an array during a quick sort.

Quick Sort

- Quick sort is O(n log n) in the average case
- $O(n^2)$ in the worst case
- Worst case can be avoided by careful choice of the pivot

Quick Sort



Concatenation of Lists

- Concatenation operation appends one list after another
  - List1; length of List 1 is N1
  - List2; length of List 2 is N2
  - concat(List1, List2); length after concatination is N1 + N2
  - Complexity of concat(…) = N2
  - concat(List1, List2, List3, …) is also allowed

- Example: List1 = [a,c,e,f] and List2 = [7,3,d]
  - concat(List1,List2) = [a,c,e,f,7,3,d]

Quick Sort: Recursive Definition

```
List quickSort(List input) {
  if(input.length() == 1)
    return input
  else {
    Element p = pickElement(input);
    return
      concat(
        quickSort(smallerHalf(input,p)),
         p,
        quickSort(biggerHalf(input,p))
      );
}
```

Time complexity

- $T_{qsort}(N) = 1$ if $N = 1$
- $T_{qsort}(N) = T_{qsort}(N1) + T(N2) + N2$
  - N1 is the length of smallerHalf
  - N2 is the length of the biggerHalf

- What is the worst case complexity?

Time complexity

- $T_{qsort}(N) = 1$ if $N = 1$
- $T_{qsort}(N) = T_{qsort}(N1) + T(N2) + N2$
  - N1 is the length of smallerHalf
  - N2 is the length of the biggerHalf

- What is the worst case complexity?
  - When N1 or N2 is equal to N-1

Worst Case Time complexity

- $T_{qsort}(N) = 1$ if $N = 1$
- $T_{qsort}(N) = T_{qsort}(1) + T(N-1) + N-1$
  - N1 is 1
  - N2 is N-1

Worst Case Complexity of Quick Sort

$T(N)$

$= T(N-1) + N$

$= T(N-2) + N-1 + N$

$= T(N-3) + N-2 + N-1 + N$

$= \ldots$

$= T(1) + 2 + 3 + \ldots + N$

$= 1 + 2 + 3 + \ldots + N$

$= N^2 / 2$

The worst case complexity of quick sort is $O(N^2)$

Quick Sort

- What is the average case complexity of Quick Sort?

Quick Sort

- What about in-place sorting?
  - Observe both quick sort and merge sort as presented are out place sorting.
  - They can be changed to in place sorting.

Quick Sort (a naïve version)

- Quick sort strategy is sound in principle, but breaks down at boundary conditions.

- What if the pivot is the largest?
  - Half of the sublist is the original list minus the pivot
  - Half of the sublist is empty
  - **Can you fix it?**

- Wastes space: does not perform in-place sorting.

Need to implement it in place

```
List quickSort(List input) { … }
```

```
void quickSort(List input) { … }
```

Partition Method

```
int partition(int[] array,
              int left, int right, int pivIndex) {
  int newPivIndex = left;
  int pivValue = array[pivIndex];
  swap(array,pivIndex,right);
  for(int i=left; i <= right-1; i++) {
    if(array[i] <= pivValue) {
      swap(array, newPivIndex, i);
      newPivIndex ++;
    }
  }
  swap(array, newPivIndex, right);
  return newPivIndex;
}
```

Quicksort Using Partition

```
int pickPivotIndex(int[] array, int left, int right) {
  . . .
}
int partition(int[] array, int left, int right) {
  . . .
}
void quickSort(int[] array, int left, int right) {
  if(right > left) {
    int p = pickPivotIndex(array, left, right);
    int newP = partition(array, left, right, p);
    quickSort(array, left, newP-1);
    quickSort(array, newP+1, right);
  }
}
void quickSort(int[] array) {
  quickSort(array, 0, array.length-1);
}
```

Readings

- Sec. 13.2