Merge Sort → divide & conquer

Sorting Fast

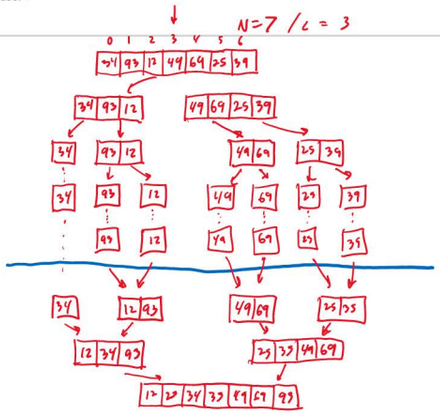
```
public class SortFast {

  public static String s(int[] arr) { return Arrays.toString(arr); }

  public static int[] combine(int[] part1, int[] part2) {
    int index1 = 0, index2 = 0;
    int[] combined = new int[part1.length + part2.length];
    while(index1 < part1.length && index2 < part2.length) {
      if(part1[index1] < part2[index2]) {
        combined[index1 + index2] = part1[index1];
        index1 += 1;
      }
      else {
        combined[index1 + index2] = part2[index2];
        index2 += 1;
      }
    }
    while(index1 < part1.length) {
      combined[index1 + index2] = part1[index1]; index1 += 1;
    }
    while(index2 < part2.length) {
      combined[index1 + index2] = part2[index2]; index2 += 1;
    }
    System.out.println(s(part1) + " + " + s(part2) + " -> " + s(combined));
    return combined;
  }

  public static int[] sortC(int[] arr) {
    if(arr.length <= 1) { return arr; }
    else {
      int[] part1 = Arrays.copyOfRange(arr, 0, arr.length / 2);
      int[] part2 = Arrays.copyOfRange(arr, arr.length / 2, arr.length);
      System.out.println(s(arr) + " -> " + s(part1) + " + " + s(part2));
      int[] sortedPart1 = sortC(part1);
      int[] sortedPart2 = sortC(part2);
      int[] sorted = combine(sortedPart1, sortedPart2);
      return sorted;
    }
  }


  public static void main(String[] args) {
    int[] result = SortFast.sortC(new int[] { 34, 93, 12, 49, 69, 25, 39 });
    System.out.println(SortFast.s(result));
  }
}
```

Draw the picture of sortC()

What is the tight bound of sortC:

Best case: $\Omega(N * \log_2(N))$

Worst case: $\Theta(N * \log_2(N))$



Splitting

$2N$

$2N - 1$

N

$N = 7$

height = 3 → $\log_2(N)$

Combining / Merging

N

N

N

height = 3 → $\log_2(N)$

$N * \log_2(N)$

$3N * \log_2(N) \rightarrow \Theta(N * \log_2(N))$

Not in-place

Needs extra memory

copyOfRange → $2N$
- new array N
- copies the values N

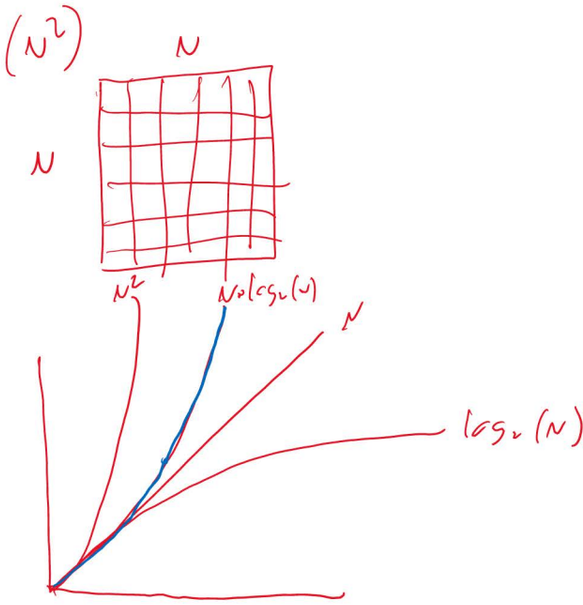
$N = 7$

$\log_2(N) = 3$

$N = 1000$

$\log_2(N) = 10$

$\Theta(N^2)$



Quick Sort

Sorting Quickly

```
public class SortQuickly {

  public static void swap(String[] array, int i1, int i2) {
    String temp = array[i1];
    array[i1] = array[i2];
    array[i2] = temp;
  }

  public static int partition(String[] array, int low, int high) {
    int pivotStartIndex = high - 1;
    String pivot = array[pivotStartIndex];
    int smallerBefore = low, largerAfter = high - 2;

    while (smallerBefore <= largerAfter) {
      if (array[smallerBefore].compareTo(pivot) < 0) {
        smallerBefore += 1;
      }
      else {
        swap(array, smallerBefore, largerAfter);
        largerAfter -= 1;
      }
    }

    swap(array, smallerBefore, pivotStartIndex);
    return smallerBefore;
  }

  public static void qsort(String[] array, int low, int high) {
    if (high - low <= 1) { return; }
    int splitAt = partition(array, low, high);
    qsort(array, low, splitAt);
    qsort(array, splitAt + 1, high);
  }

  public static void sortD(String[] array) {
    qsort(array, 0, array.length);
  }

  public static void main(String[] args) {
    String[] str = {"f", "b", "a", "e", "d", "c" };
    int[] result = SortQuickly.sortD(str);
    System.out.println(Arrays.deepToString(result));
  }
}
```

Draw the picture of sortD()

What is the tight bound of sortD:

Best case:

Worst case:

$\Theta(1)$ in-place

low = 0
high = 6

pivot = 'c'
sb = 0+1/2
la = 4-3-1