Algorithms and Data Structure 5

Yiping Deng

March 12, 2018

Problem 1

a)

File 1: QuickSortVariant.scala

```
object QuickSortVariant {
  def swap[T](arr: Array[T], i1: Int, i2: Int):Unit = {
    val tmp = arr(i1)
    arr(i1) = arr(i2)
    arr(i2) = tmp
  }

  def partition[T](arr: Array[T], first: Int, last: Int)(implicit ord: Ordering[T]):(Int, Int) = {
    //make a three way partition
    //return the index of first and second pivot
    //just don't do stupid thing with this three way partition

    if ((last - first) >= 3) {
      //choose first two as pivot
      if (ord.gt(arr(first), arr(first + 1))) {
        swap(arr, first, first + 1)
      }
      //put it at front and back
      swap(arr, first + 1, last)

      val p = arr(first)
      val q = arr(last)

      var l = first + 1
      var g = last - 1
      var k = l

      //maintain the loop invariant
      while (k <= g) {
        if (ord.lt(arr(k), p)) {
          //move the lower up, add one
          swap(arr, k, l)
          l = l + 1
        } else if (ord.gteq(arr(k), q)) {
          //move the bound for last partition in the right position
          while (ord.gt(arr(g), q) && k < g)
            g = g - 1

          swap(arr, k, g) //add one more
          g = g - 1

          if (ord.lt(arr(k), p)) {
```

```
          //if the new swapped can be move to even lower categories
          swap(arr, k, l)
          l = l + 1
        }
      }
      k = k + 1 //move the index
    }
    //fix the index
    l = l - 1
    g = g + 1

    //move the pivot in place
    swap(arr, first, l)
    swap(arr, last, g)

    //return the pivot
    (l, g)
  } else if (first == last - 1) {
    //two element partition
    if (ord.gt(arr(first), arr(last)))
      swap(arr, first, last)
    (first, last)
  } else if (first == last - 2) {
    //three element partition
    if (ord.gt(arr(first), arr(first + 1)))
      swap(arr, first, first + 1)
    if (ord.gt(arr(first + 1), arr(last)))
      swap(arr, first + 1, last)
    if (ord.gt(arr(first), arr(first + 1)))
      swap(arr, first, first + 1)
    (first, last)
  } else if(first == last){
    (first, first)
  } else {
    //should never happen
    throw new Exception
  }
}



def helper[T](arr: Array[T], first: Int, last: Int)(implicit ord: Ordering[T]):Unit = {
  //sorting from the first to the last, including the last

  if(first < last){
    val (p1, p2) = partition(arr, first, last)
    helper(arr, first, p1 - 1)
    helper(arr, p1 + 1, p2 - 1)
    helper(arr, p2 + 1, last)
  }
  //put this two pivot in the right order
}

def apply[T](arr: Array[T])(implicit ord: Ordering[T]):Unit = {
  // the main algorithm
  // just call the helper function
  helper(arr, 0, arr.length - 1)
}
```

```
}
```

Test case
File 2: QuickSortTest.scala

```
import org.scalatest.FlatSpec

class QuickSortTest extends FlatSpec {
  "A Partition algorithm" should "divide the array properly in all case" in {
    val arr = Array(2, 1)
    val (t1, t2) = QuickSortVariant.partition(arr, 0, 1)
    assert(arr(t1) < arr(t2))

    val arr1 = Array(2,1, -1, 3, 4, 0)
    val (t3, t4) = QuickSortVariant.partition(arr1, 0, arr1.length - 1)
    println(arr1.mkString(","))
    println(t3)
    println(t4)
    assert(t3 == 2)
    assert(t4 == 3)

    val r = scala.util.Random
    val rand_arr = (1 to 50).map{ _ => r.nextInt(100)}.toArray
    val (p1, p2) = QuickSortVariant.partition(rand_arr, 0, rand_arr.length - 1)

    val sub_arr1 = rand_arr.slice(0, p1)
    println(sub_arr1.mkString(" "))
    val sub_arr2 = rand_arr.slice(p1, p2)
    println(sub_arr2.mkString(" "))
    val sub_arr3 = rand_arr.slice(p2, rand_arr.length)
    println(sub_arr3.mkString(" "))

  }
  "A sorting algorithm" should "sort everything in place" in {
    val r = scala.util.Random
    val arr = (1 to 50) map { _ => r.nextInt(100)} toArray
    val arr1 = arr.clone().sorted //make a copy of the array and sort
    val arr2 = arr.clone()

    QuickSortVariant(arr)

    assert(arr.deep == arr1.deep)

    QuickSortVariantRandom(arr2)

    assert(arr2.deep == arr1.deep)
  }
}
```

b)

Best Case: Best case will lead to equal partition in all recursive calls. Every recursions will make 3 recursive calls on a $n/3$ data set. Partition iterates through the element once, so it's complexity is $O(n)$. Thus we can write the recurrences, and solve it.

$$T(n) = 3T(n) + O(n)$$
$$T(n) = O(n \log(n))$$

Hence, the best case is $O(n \log(n))$
Worst Case: Worst case will lead to partition that place the two pivots at the beginning and at the end. The first partition contains 0 element, and the second partition contains $n-2$ elements, and the third partition contains 0 elements. Essentially, it just put the two pivot at the right position on every recursive call, nothing else. Partition iterates through the element once, so it's complexity is $O(n)$. Thus, we can write the recurrence as following and solve it

$$T(n) = T(n-2) + O(n)$$
$$T(n) = O(n^2)$$

Hence, the worst case is $O(n^2)$.

c)

Let's modify our partition a bit, and change nothing else.

File 1: QuickSortVariantRandom.scala

```scala
object QuickSortVariantRandom {
  import QuickSortVariant.swap

  val r = scala.util.Random
  def randPair(max: Int): (Int, Int) = {
    if(max <= 1)
      (0, 1)
    else {
      val r1 = r.nextInt(max + 1)
      val r2 = r.nextInt(max + 1)
      if(r1 == r2)
        randPair(max)
      else
        (r1, r2)
    }
  }
  def partition[T](arr: Array[T], first: Int, last: Int)(implicit ord: Ordering[T]):(Int, Int) = {
    // this is a randomized pivots variant
    //make a three way partition
    //return the index of first and second pivot
    //just don't do stupid thing with this three way partition

    if ((last - first) >= 3) {
      //choose two pivot and put it in the front
      val (r1, r2) = randPair(last - first)
      swap(arr, first + r1, first)
      swap(arr, first + r2, first + 1)


      if (ord.gt(arr(first), arr(first + 1))) {
        swap(arr, first, first + 1)
      }
      //put it at front and back
      swap(arr, first + 1, last)

      val p = arr(first)
      val q = arr(last)

      var l = first + 1
      var g = last - 1
      var k = l
```

```
      //maintain the loop invariant
      while (k <= g) {
        if (ord.lt(arr(k), p)) {
          //move the lower up, add one
          swap(arr, k, l)
          l = l + 1
        } else if (ord.gteq(arr(k), q)) {
          //move the bound for last partition in the right position
          while (ord.gt(arr(g), q) && k < g)
            g = g - 1

          swap(arr, k, g) //add one more
          g = g - 1

          if (ord.lt(arr(k), p)) {
            //if the new swapped can be move to even lower categories
            swap(arr, k, l)
            l = l + 1
          }
        }
        k = k + 1 //move the index
      }
      //fix the index
      l = l - 1
      g = g + 1

      //move the pivot in place
      swap(arr, first, l)
      swap(arr, last, g)

      //return the pivot
      (l, g)
    } else if (first == last - 1) {
      //two element partition
      if (ord.gt(arr(first), arr(last)))
        swap(arr, first, last)
      (first, last)
    } else if (first == last - 2) {
      //three element partition
      if (ord.gt(arr(first), arr(first + 1)))
        swap(arr, first, first + 1)
      if (ord.gt(arr(first + 1), arr(last)))
        swap(arr, first + 1, last)
      if (ord.gt(arr(first), arr(first + 1)))
        swap(arr, first, first + 1)
      (first, last)
    } else if(first == last){
      (first, first)
    } else {
      //should never happen
      throw new Exception
    }
  }
  def helper[T](arr: Array[T], first: Int, last: Int)(implicit ord: Ordering[T]):Unit = {
    //sorting from the first to the last, including the last

    if(first < last){
      val (p1, p2) = partition(arr, first, last)
```

```
      helper(arr, first, p1 - 1)
      helper(arr, p1 + 1, p2 - 1)
      helper(arr, p2 + 1, last)
    }
    //put this two pivot in the right order
  }

  def apply[T](arr: Array[T])(implicit ord: Ordering[T]):Unit = {
    // the main algorithm
    // just call the helper function
    helper(arr, 0, arr.length - 1)
  }
}
```

Problem 2

a)

Proof: We start by simpify the LHS

$$\begin{aligned}
LHS &\leq \sum_{k=1}^{n-1} k\lg(k) \leq \sum_{k=1}^{\lfloor n/2 \rfloor} k\lg(k) + \sum_{k=\lfloor n/2 \rfloor+1}^{n-1} k\lg(k) \\
&\leq lg(\lfloor n/2 \rfloor) \sum_{k=1}^{\lfloor n/2 \rfloor} k + \lg(n) \sum_{k=\lfloor n/2 \rfloor+1}^{n-1} k\lg(k) \\
&\leq (\lg(n) - 1) \sum_{k=1}^{\lfloor n/2 \rfloor} k + \lg(n) \sum_{k=\lfloor n/2 \rfloor+1}^{n-1} k\lg(k) \\
&\leq \lg(n) \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor} k
\end{aligned}$$

Lemma: $\sum_{k=1}^{n} = \frac{n(n+1)}{2}$
Using induction:
Basis: $n = 1$

$$\sum_{k=1}^{n} k = 1 = \frac{1 \cdot (1+1)}{2} = 1$$

Inductive step: Assume it holds for n

$$\begin{aligned}
\sum_{k=1}^{n+1} k &= \sum_{k=1}^{n} k + n + 1 \\
&= \frac{n(n+1)}{2} + \frac{2n+2}{2} \\
&= \frac{n^2+3n+2}{2} \\
&= \frac{(n+1)(n+2)}{2}
\end{aligned}$$

Using this to simplify the formula, we have

$$\begin{aligned}
LHS &\leq \lg(n)\frac{n(n-1)}{2} - \frac{n/2 \cdot (n/2+1)}{2} \\
&= \lg(n)\frac{n^2/2}{2} - \frac{n}{2}\lg(n) - \frac{n^2}{8} - \frac{n}{4}\lg(n) \\
&\leq \frac{1}{2}n^2\lg(n) - \frac{1}{8}n^2
\end{aligned}$$

b)

Assumption: Uniform distribution of permutation.
Proof: define random variable

$$X_k := \begin{cases} 1 & \text{if partition at index k} \\ 0 & \text{if not} \end{cases}$$

Due to uniform distribution, we have

$$E(X_k) = 1/n$$

Thus,

$$T(n) := \begin{cases} T(0) + T(n-1) + \Theta(n) & \text{if partition at index 0} \\ T(1) + T(n-2) + \Theta(n) & \text{if partition at index 1} \\ & ... \\ & ... \\ T(n-1) + T(0) + \Theta(n) & \text{if partition at index } n-1 \end{cases}$$

Taking expectation on both side and use linearity of expectation.
Also merge two similar sum. Merge the case $k = 0, k = 1$ into the term $\Theta(n)$

$$\begin{aligned} E(T(n)) &= E(\sum_{k=0}^{n-1} X_k(T(k) + T(n-k-1)) +) \\ &= 1/n \sum_{k=0}^{n-1} E(T(k)) + 1/n \sum_{k=0}^{n-1} E(T(n-k-1)) + 1/n \sum_{k=0}^{n-1} \Theta(n) \\ &= \frac{2}{n} \sum_{k=1}^{n-1} E(T(k)) + \Theta(n) \end{aligned}$$

Base case: $k = 2$, it is quite obvious since c can be arbitrary big.
Inductive case:(strong induction)
Assume the argument holds for any $n \leq m - 1$

$$\begin{aligned} E(T(m)) &\leq \frac{2}{n} \sum_{k=2} m - 1c \cdot k \lg(k) + \Theta(n) \\ &\leq \frac{2c}{n}(\frac{1}{2}n^2 \lg(n) - \frac{1}{8}n^2) \\ &\leq cn \lg(n) - \frac{cn}{4} + \Theta(n) \\ &\leq cn \lg(n) \text{ for arbitrary c} \end{aligned}$$

Problem 3:

Part 1: Upper bound.

$$\begin{gathered} n! \leq n^n \implies \\ \lg(n!) \leq \lg(n^n) = n \lg(n) \end{gathered}$$

Part 2: Lower bound.

$$\begin{gathered} (n!)^2 \geq n^n \implies \\ \lg((n!)^2) = 2 \lg(n!) \geq n \lg(n) \implies \\ \lg(n!) \geq \frac{1}{2} n \lg(n) \end{gathered}$$

This directly implies that

$$\lg(n!) = \Theta(n \lg(n))$$