CS5800: Algorithms — Spring '21 — Virgil Pavlu

Homework 2A
Due : Gradescope

Name: Zerun Tian

Instructions:

- Make sure to put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- Please review the grading policy outlined in the course information page.
- You must also write down with whom you worked on the assignment. If this changes from problem to problem, then you should write down this information separately with each problem.
- Problem numbers (like Exercise 3.1-1) are corresponding to CLRS 3^{rd} edition. While the 2^{nd} edition has similar problems with similar numbers, the actual exercises and their solutions are different, so make sure you are using the 3^{rd} edition.

1. ***(5 points)*** *Exercise 6.1-4, explain why.*

Where in a max-heap might the smallest element reside, assuming that all elements are distinct?

Solution:

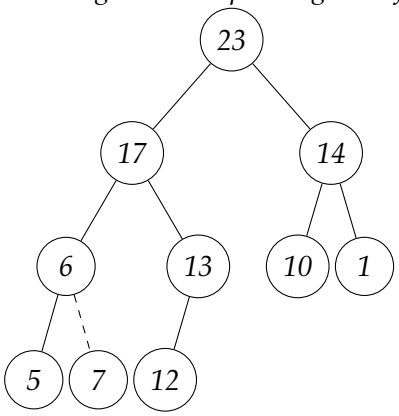
In the tree representation of a max-heap, the smallest element resides in one of the leaf nodes. In the array representation, the smallest element appears in $A[\lfloor n/2 \rfloor + 1 :]$*, from index* $\lfloor n/2 \rfloor + 1$ *to the last index (inclusion). We can prove this by contradiction. Let's assume that the smallest element exists in a non-leaf node. Because all elements are distinct, the values in children of the non-leaf node, containing the smallest value, must be greater, which breaks the max-heap property. Our assumption leads to an invalid max-heap, so we showed that the smallest element resides in a leaf node.*

2. ***(5 points)*** *Exercise 6.1-6, explain why.*

Is the array with values $[23, 17, 14, 6, 13, 10, 1, 5, 7, 12]$ *a max-heap?*

Solution:

Drawing the corresponding tree of the above values, we can easily spot any problems.

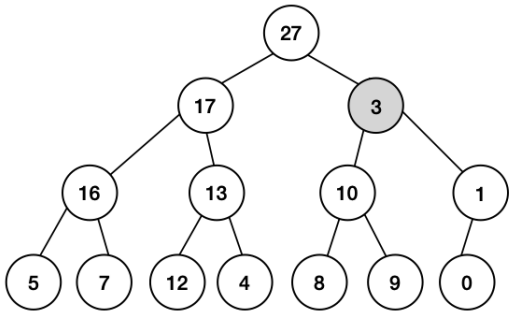


We notice that the node with value 6 is smaller than the right child with value 7. This break the max-heap property, so it's not a max-heap.

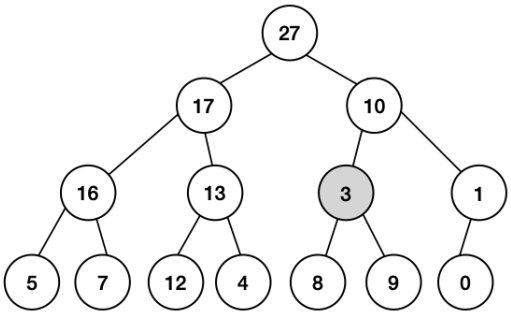
3. ***(10 points)*** *Exercise 6.2-1.*

Illustrate the operation of MAX-HEAPIFY(A, 3) on the array $A = [27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0]$.

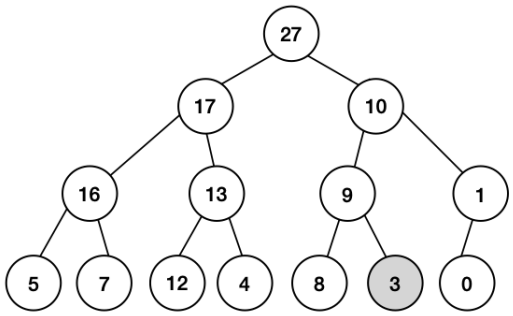
Solution:



(a)



(b)



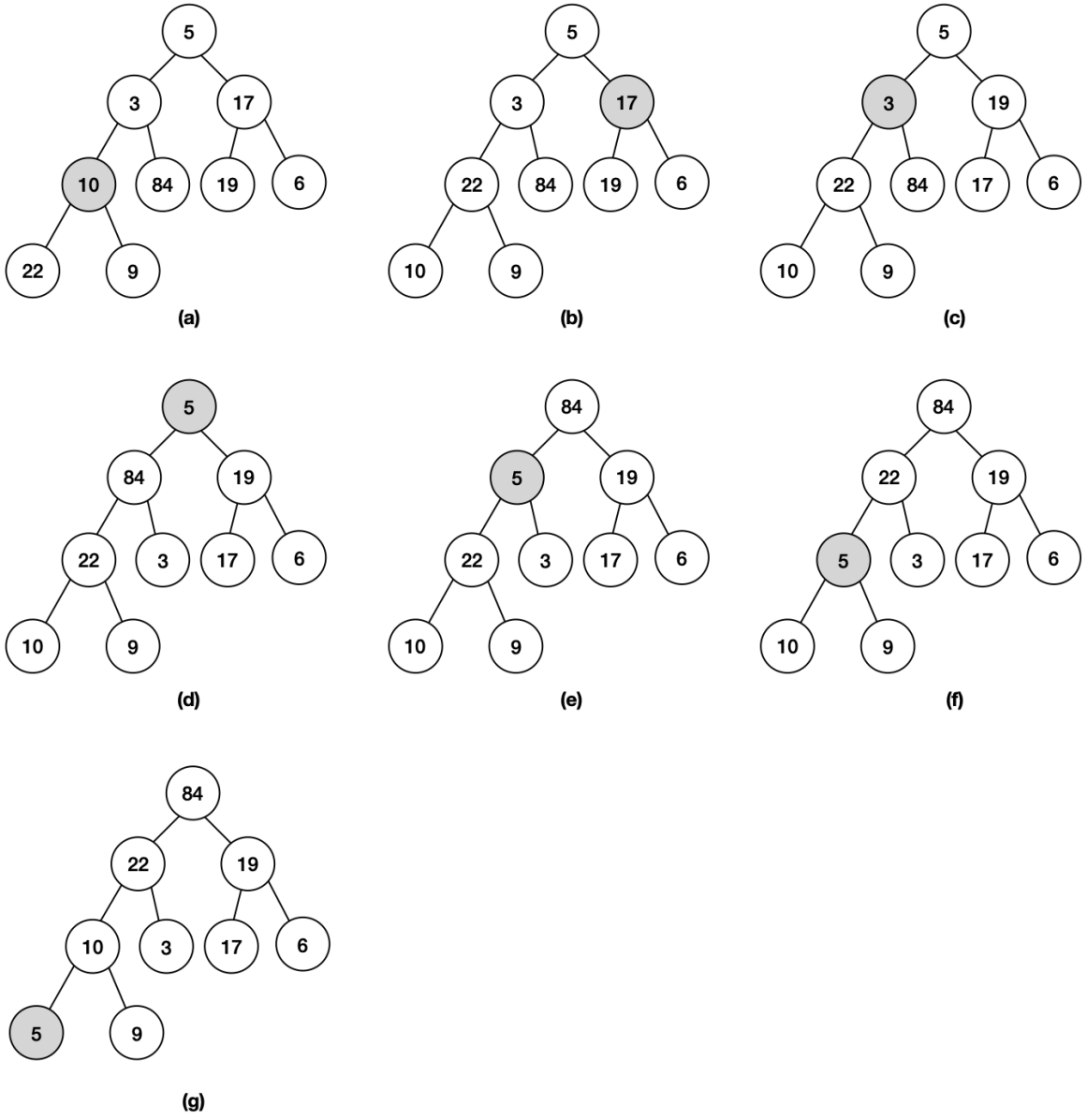
(c)

In the initial configuration (a), the node at index 3 has value 3 which is not greater than its direct left child of value 10. In (b), we exchange the value 10 with 3. Then, it enters the recursive call, which compares 3 with direct children out of which the right child has a greater value of 9. Performing the same exchange operation, we end up with the configuration (c) which terminates.

4. ***(10 points)*** *Exercise 6.3-1.*

Illustrate the operation of BUILD-MAX-HEAP on the array $A = [5, 3, 17, 10, 84, 19, 6, 22, 9]$.

Solution:



In the initial configuration (a), we start from the gray node of value 10. It compares with its direct children among which the left child of value 22 is greater. The value 10 is swapped with 22, and we move on to the next gray node of value 17. In configuration (b), we find 17 is not greater than its direct left child 19, running MAX-HEAPIFY results in (c). In the next iteration, the gray node of value 3 is less than both children, among which the right child of value 84 is larger, which results in (d). In (e), the gray node of value 5 is not greater its left child 22. Running MAX-HEAPIFY results in (f). Finally, the max-heap is built completely after the value 5 is swapped with 10 which is greater.

5. ***(15 points)*** *Problem 6-2.*

1. *How would you represent a d-ary heap in an array?*

   Solution:

   Suppose we are using 1-based indexing. To find the k-th child of the element i, we can compute $(i-1)d+k+1$ to get its exact index in the array. The parent of the child i can be found using the formula, $\lfloor (i-2+d)/d \rfloor$. As such, we can represent the parent-child relationship of a d-ary heap in an array.

2. *What is the height of a d-ary heap of n elements in terms of n and d?*

   Solution:

   Suppose that the heap is full, meaning the last layer of the tree is filled with leaves. The total number of nodes in the heap, n, can be written as a geometric series. We know the sum of such sequence can be calculated using the closed-form formula — $(1-d^k)/(1-d)$ where k is the number of layers of the tree. The height of the tree is just $k-1$. Let's solve k,

$$\begin{aligned}
\frac{1-d^k}{1-d} &= n \\
1-d^k &= n-dn \\
d^k &= dn-n+1 \\
k &= \log_d^{dn-n+1}
\end{aligned}$$

   The height of such heap is $\log_d^{dn-n+1}-1 = \log_d^{n-n/d+1/d}$. On the other hand, the last layer of the same height tree can contain as little as one leaf (all other leaf nodes are one-layer above). In this case, we can express n to be $(1-d^{k-1})/(1-d)+1$.

$$\begin{aligned}
\frac{1-d^{k-1}}{1-d}+1 &= n \\
\frac{1-d^{k-1}}{1-d} &= n-1 \\
1-d^{k-1} &= n-dn-1+d \\
d^{k-1} &= dn-n+2-d \\
k-1 &= \log_d^{dn-n+2-d}
\end{aligned}$$

   In this case, the height is $\log_d^{dn-n+2-d}$. We observe that the height of the tree is bounded by $\log_d^n$.

3. *Give an efficient implementation of EXTRACT-MAX in a d-ary max-heap. Analyze its running time in terms of d and n.*

Solution:

Suppose we are working with 1-based indexing. Before we start, let's define two primitive helper functions PARENT *and* CHILD *using the answer to problem 5.1.*

```
1: function PARENT(i)
2:     return floor((i − 2 + d)/d)
```

```
1: function CHILD(i, k)                          ▷ the k-th child of node i
2:     return (i − 1)d + k + 1
```

The EXTRACTMAX *function is exactly the same as it is for the binary heap.*

```
1: function EXTRACTMAX(A)
2:     if A.heap-size < 1 then
3:         throw error "heap underflow"
4:     max = A[1]
5:     A[1] = A[A.heap-size]
6:     A.heap-size = A.heap-size − 1
7:     MAXHEAPIFY(A, 1)
8:     return max
```

What differs is the MAXHEAPIFY *function.*

```
1: function MAXHEAPIFY(A, i)
2:     largest = i
3:     for k = 1, k <= d, k++ do                 ▷ find a child that has the largest value
4:         child = CHILD(A, i, k)
5:         if child ≤ A.heap-size and A[child] > A[largest] then
6:             largest = child
7:     if largest ≠ i then                       ▷ swap the largest child with the value of the current node i
8:         tmp = A[largest]
9:         A[largest] = A[i]
10:        A[i] = tmp
11:        MAXHEAPIFY(A, largest)                ▷ call recursively on the node being updated
```

For the MAXHEAPIFY *function, the depth of the recursion is at most the height of the heap, which is bounded by* $\log_d^n$*. In every call, constant amount of work are executed d times in the for loop at line 3. In total, the running time is* $O(d \log_d^n)$*.*

4. *Give an efficient implementation of INSERT in a d-ary max-heap. Analyze its running time in terms of d and n.*

Solution:

```
1: function INSERT(A, val)
2:     A.heap-size = A.heap-size + 1                ▷ create a leaf and add the value there
3:     A[A.heap-size] = val
4:     i = A.heap-size
5:     while i > 1 and A[PARENT(i)] < A[i] do  ▷ move the node i up if it's greater than its parent
6:         tmp = A[i]
7:         A[i] = A[PARENT(i)]
8:         A[PARENT(i)] = tmp
9:         i = PARENT(i)
```

Operations prior to the while loop takes constant time. What's inside the while loop take constant time, and the loop executes at most "the height of the heap" times. Thus, the running time is $O(\log_d^n)$.

5. *Give an efficient implementation of INCREASE-KEY(A, i, k), which flags an error if* $k < A[i]$*, but otherwise sets* $A[i] = k$ *and then updates the d-ary max-heap structure appropriately. Analyze its running time in terms of d and n.*

Solution:

```
1: function INCREASEKEY(A, i, k)
2:     if k < A[i] then
3:         throw error "the new value k is not greater than the current value"
4:     A[i] = k
5:     while i > 1 and A[PARENT(i)] < A[i] do
6:         tmp = A[i]
7:         A[i] = A[PARENT(i)]
8:         A[PARENT(i)] = tmp
9:         i = PARENT(i)
```

Similar to the last problem, operations prior to the while loop takes constant time. What's inside the loop takes constant time, and the loop executes at most $\log_d^n$ *times, which is the depth of the heap. Thus, the running time is* $O(\log_d^n)$.

6. ***(5 points)*** *Exercise 7.2-1.*

Use the substitution method to prove that the recurrence $T(n) = T(n-1) + \Theta(n)$ has the solution $T(n) = \Theta(n^2)$, as claimed at the beginning of Section 7.2.

Solution:

We want to show that $c_1 n^2 \leq T(n) \leq c_2 n^2$.

- First, let's prove the lower bound $T(n) \geq c_1 n^2$ via induction.

Let's make an inductive hypothesis, $T(n-1) \geq c_1(n-1)^2$.

$$\begin{aligned} T(n) &= T(n-1) + \Theta(n) \\ &\geq c_1(n-1)^2 + c_3 n \\ &= c_1 n^2 - 2c_1 n + c_1 + c_3 n \\ &= c_1 n^2 - (2c_1 - c_3)n + c_1 \end{aligned}$$

We want the expanded expression to be greater than or equal to $c_1 n^2$,

$$\begin{aligned} T(n) \geq c_1 n^2 - (2c_1 - c_3)n + c_1 &\overset{?}{\geq} c_1 n^2 \\ (c_3 - 2c_1)n + c_1 &\overset{?}{\geq} 0 \end{aligned}$$

We can choose $c_1 = 1$ and $c_3 = 2$, so we have $1 \geq 0$. The lower bound is proved to be correct via induction.

- Second, let's show the upper bound $T(n) \leq c_2 n^2$ is correct.

Let's make an inductive hypothesis, $T(n-1) \leq c_2(n-1)^2$.

$$\begin{aligned} T(n) &= T(n-1) + \Theta(n) \\ &\leq c_2(n-1)^2 + c_4 n \\ &= c_2 n^2 - 2c_2 n + c_2 + c_4 n \\ &= c_2 n^2 - (2c_2 - c_4)n + c_2 \end{aligned}$$

We want the expression to be less than or equal to $c_2 n^2$,

$$\begin{aligned} T(n) \leq c_2 n^2 - (2c_2 - c_4)n + c_2 &\overset{?}{\leq} c_2 n^2 \\ (c_4 - 2c_2)n + c_2 &\overset{?}{\leq} 0 \\ c_2 &\overset{?}{\leq} (2c_2 - c_4)n \end{aligned}$$

Based on the above inequality, we can choose c_2, c_4 such that $2c_2 \geq c_4$ and $n \geq c_2/(2c_2 - c_4)$. Let's choose $c_2 = 1$ and $c_4 = 1$. Plugging them into the equation, we have $1 \leq n$. Since n could be arbitrarily large, we proved the upper bound is correct by induction. Thus, $T(n) = \Theta(n^2)$.

7. *(5 points)* *Exercise 7.2-2, explain why.*

What is the running time of QUICKSORT when all elements of array A have the same value?

Solution:

When all elements are the same, the partition is unbalanced at every recursive call in which one partition has zero element, and the other has the rest. This is the worst case scenario for quicksort described in the page 175 of the CLRS book. The corresponding recurrence is $T(n) = T(n-1) + T(0) + \Theta(n) = T(n-1) + \Theta(n)$*, which has been shown to be quadratic* $T(n) = \Theta(n^2)$ *in problem 6.*

8. ***(5 points)*** *Exercise 7.2-3.*

Show that the running time of QUICKSORT is $\Theta(n^2)$ when the array A contains distinct elements and is sorted in decreasing order.

Solution:

In quicksort, elements are shuffled into partitions. Let's say the elements smaller than or equal to the pivot are in the "small" partition, and the elements greater than pivot are in the "big" partition for the sake of simplicity.

The pivot is selected as the right most element bounded by the left and right indices in each recursive call. From the beginning, because the array is already sorted in descending order, elements from the left are all greater than the pivot, so the "small" partition has size 0 and the "big" partition has size $n-1$. The pivot (the smallest value) will be swapped with the first element of the "big" partition (the largest value). In the next call, the new pivot ends up being the largest value of the array. Again, the partition is extremely unbalanced with the "small" partition of size $n-2$ and the "big" partition of size 0.

So on so forth, we realize that the partition is always unbalanced. Similar to problem 7, the recurrence is $T(n) = T(n-1) + T(0) + \Theta(n) = T(n-1) + \Theta(n)$, which has been shown to be quadratic $T(n) = \Theta(n^2)$ in problem 6.

9. ***(15 points)*** *Exercise 7.4-1.*

Show that in the recurrence $T(n) = \max_{0 \le q \le n-1}(T(q) + T(n-q-1)) + \Theta(n)$, $T(n) = \Omega(n^2)$.

Solution:

Similar to how the book (CLRS 7.4.1) proved the upper bound - $O(n^2)$, the worst-case running time of the recurrence of quicksort, we can prove the lower bound $\Omega(n^2)$ using the substitution method.

Let's make a strong inductive hypothesis that $T(m) \ge cm^2$ for all $m < n$. Substituting this hypothesis into the recurrence above, we have,

$$\begin{aligned} T(n) &= \max_{0 \le q \le n-1}(T(q) + T(n-q-1)) + \Theta(n) \\ &\ge \max_{0 \le q \le n-1}(cq^2 + c(n-q-1)^2) + \Theta(n) \\ &= c \cdot \max_{0 \le q \le n-1}(q^2 + (n-q-1)^2) + \Theta(n) \end{aligned}$$

Let's find the parameter q that let the expression $f(n,q) = q^2 + (n-q-1)^2$ achieve a maximum.

$$\begin{aligned} f(n,q) = q^2 + (n-q-1)^2 &= q^2 + n^2 - nq - n - nq + q^2 + q - n + q + 1 \\ &= n^2 + 2q^2 - 2nq - 2n + 2q + 1 \end{aligned}$$

The first derivative w.r.t. q is $\frac{\partial f}{\partial q} = 4q - 2n + 2$. The second derivative w.r.t. q is that $\frac{\partial^2 f}{\partial q^2} = 4$, which is positive. This tells us that a maximum can be found at either endpoint of the bound. Plugging either $q = 0$ or $q = n-1$ into $f(n,q)$, we have $(n-1)^2$. This tells us that $\max_{0 \le q \le n-1}(q^2 + (n-q-1)^2) = (n-1)^2 = n^2 - 2n + 1$. Let's substitute that back to the recurrence,

$$\begin{aligned} T(n) &\ge c \cdot \max_{0 \le q \le n-1}(q^2 + (n-q-1)^2) + \Theta(n) \\ &= c(n^2 - 2n + 1) + \Theta(n) \\ &= cn^2 - c(2n-1) + \Theta(n) \\ &\overset{?}{\ge} cn^2 \end{aligned}$$

We want $cn^2 - c(2n-1) + \Theta(n) \overset{?}{\ge} cn^2$, so simplifying it,

$$\begin{aligned} -c(2n-1) + \Theta(n) &\overset{?}{\ge} 0 \\ -2nc + c + c_1 n &\overset{?}{\ge} 0 \end{aligned}$$

We can choose $c = 1$ and $c_1 = 2$, which gives us $-2n + 1 + 2n = 1 \ge 0$. We successfully showed that $T(n) \ge cn^2$, then we know $T(n) = \Omega(n^2)$.

10. ***(15 points)*** *Exercise 7.4-2.*

Show that quicksort's best-case running time is $\Omega(n \lg n)$.

Solution:

On the contrary to its worst-case's recurrence, quicksort's best-case can be written as,

$$T(n) = \min_{0 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n)$$

Let's prove this using the substitution method. We make a hypothesis that $T(m) \ge c(m \lg m + 2m)$ *for all* $m < n$ *and some constant* c. *Substituting that back into the recurrence, we get,*

$$\begin{aligned}
T(n) &= \min_{0 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n) \\
&\ge \min_{0 \le q \le n-1} (c(q \lg q + 2q) + c((n-q-1)\lg(n-q-1) + 2(n-q-1))) + \Theta(n) \\
&= c \cdot \min_{0 \le q \le n-1} (q \lg q + 2q + (n-q-1)\lg(n-q-1) + 2(n-q-1)) + \Theta(n)
\end{aligned}$$

Let's find a minimum of the function $f(n,q) = q \lg q + 2q + (n-q-1)\lg(n-q-1) + 2(n-q-1)$ *by taking the first derivative and set to 0.*

$$\begin{aligned}
\frac{\partial f}{\partial q} &= 1 \cdot \lg q + q \cdot \frac{1}{\ln 2} \cdot \frac{1}{q} + 2 - \lg(n-q-1) - (n-q-1) \cdot \frac{1}{\ln 2} \cdot \frac{1}{n-q-1} - 2 \\
&= \lg q + \frac{1}{\ln 2} + 2 - \lg(n-q-1) - \frac{1}{\ln 2} - 2 \\
&= \lg q - \lg(n-q-1)
\end{aligned}$$

Setting the first derivative to 0, we have,

$$\begin{aligned}
\lg q - \lg(n-q-1) &= 0 \\
\lg \frac{q}{n-q-1} &= 0 \\
\frac{q}{n-q-1} &= 1 \\
q &= n-q-1 \\
q &= \frac{n-1}{2}
\end{aligned}$$

The function $f(n,q)$ *reaches a minimum when* $q = (n-1)/2$. *Substituting* q, *we have,*

$$\begin{aligned}
T(n) &= c \cdot \min_{0 \le q \le n-1} (q \lg q + 2q + (n-q-1)\lg(n-q-1) + 2(n-q-1)) + \Theta(n) \\
&= c(\frac{n-1}{2} \lg \frac{n-1}{2} + n - 1 + \frac{n-1}{2} \lg \frac{n-1}{2} + n - 1) + \Theta(n) \\
&= c((n-1)\lg \frac{n-1}{2} + 2n - 2) + \Theta(n) \\
&= c((n-1) \cdot (\lg(n-1) - 1) + 2n - 2) + \Theta(n) = c((n-1)\lg(n-1) + n - 1) + \Theta(n) \\
&= c(n-1)\lg(n-1) + cn - c + \Theta(n) = cn(\lg(n-1) + 1) - c\lg(n-1) - c + \Theta(n) \\
&= cn\lg(2n-2) - c\lg(n-1) - c + \Theta(n) \overset{?}{\ge} cn \lg n
\end{aligned}$$

If we pick a c *that is large enough and for a large* n, $cn\lg(2n-2)$ *term dominates* $-c\lg(n-1) - c + \Theta(n)$. *We showed* $T(n) = \Omega(n \lg n)$. *Thus, quicksort's best-case running time is* $\Omega(n \lg n)$.

11. ***(10 points)*** *Exercise 7.4-3.*

Show that the expression $q^2 + (n - q - 1)^2$ *achieves a maximum over* $q = 0, 1, \cdots, n - 1$ *when* $q = 0$ *or* $q = n - 1$.

Solution:

To find a maximum of the expression, we first check the sign of the second derivative.

$$\begin{aligned} f(n, q) = q^2 + (n - q - 1)^2 &= q^2 + n^2 - nq - n - nq + q^2 + q - n + q + 1 \\ &= n^2 + 2q^2 - 2nq - 2n + 2q + 1 \end{aligned}$$

The first derivative w.r.t. q *is that* $\frac{\partial f}{\partial q} = 4q - 2n + 2$. *The second derivative w.r.t.* q *is that* $\frac{\partial^2 f}{\partial q^2} = 4$, *which is positive. This means that a critical point will be a minimum, and more importantly, either endpoint of the bound will be a maximum. When* $q = 0$, *the expression becomes* $(n - 1)^2$, *and when* $q = n - 1$, *the expression is also* $(n - 1)^2$. *Thus, when* $q = 0$ *or* $q = n - 1$, *the expression reaches a maximum.*

12. ***(Extra credit 10 points)*** *Problem 6-3.*