Missouri University of Science & Technology | Department of Computer Science
**Spring 2023** | **CS 2500: Algorithms (Sec: 102)**

# Solutions to Homework 2

**Name:** *Sid Nadendla* | **Email:** *nadendla@mst.edu*

## Problem 1. Workflow of Heapsort and Quicksort *25 points*

Demonstrate HEAP-SORT and QUICK-SORT iterations for both the following arrays:
(i) $A_1 = \{2, 6, 4, 3, 1, 5\}$, and (ii) $A_2 = \{1, 5, 2, 3, 0, 2, 2, 1, 4, 5\}$.

**SOLUTION:**
This problem considers two sorting algorithms, namely `Heap-Sort` and `Quick-Sort`, whose pesudocodes are given below:

```
HEAPSORT(A)
BUILD-MAX-HEAP(A)
for i = ⌊A.length⌋ downto 2
    Swap A[1] and A[i]
    A.heap-size = A.heap-size − 1
    MAX-HEAPIFY(A, 1)
```

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
```

(i) Let us consider the first array $A_1 = \{2, 6, 4, 3, 1, 5\}$. The various stages of `Heap-Sort`($A_1$) are demonstrated in Figure 1. Similarly, the various stages of `Quick-Sort`($A_1$) are demonstrated in Figure 2.

(ii) The various stages of `Heap-Sort`($A_2$) and `Quick-Sort`($A_2$) can be demonstrated similar to that shown in (i).

## Problem 2. Empirical Analysis of Heapsort and Quicksort *25 points*

Implement HEAP-SORT (Page 170 with supporting functions in Pages 165, 167, all in *CLRS*) and QUICK-SORT (Page 183, *CLRS*) in Python, and validate its average run-time performance (similar to Problem 2 in Homework 1).

**SOLUTION:** Code will be provided as a Jupyter notebook, along with these solutions. □

## Problem 3. Modified Quicksort *25 points*

Traditional quicksort routine chooses a pivot $q$ such that $A[p : q-1] \leq A[q] \leq A[q+1, r]$. Instead, present an analysis when the quicksort algorithm partitions the array $A[p : r]$ into three parts using two pivots $q_1$ and $q_2$ such that $A[p : q_1 - 1] \leq A[q_1] = \cdots = A[q_2] \leq A[q_2 + 1 : r]$.
(Hint: Assume that the entries in $A$ are picked from $\{1, \cdots, m\}$, where $m < n$.)

**SOLUTION:**

This modified quicksort is called threeway quicksort, and is known to improve the run-time of the traditional quicksort by a significant factor when the input array contains repeated entries.

Given two pivots $q_1$ and $q_2$ such that the array $A$ satisfies $A[p : q_1 - 1] \leq A[q_1] = \cdots = A[q_2] \leq A[q_2 + 1 : r]$, the problem reduces to sorting the left sub-array $L = A[p : q_1 - 1]$ and the right sub-array $R = A[q_2 + 1 : r]$. Note that the size of $L$ and $R$ are $q_1 - 1$ and $n - q_2$ respectively. If $T(n)$ represents the run-time of threeway quicksort on input array $A$, then its run-time recursion is given by

$$T(n) = T(q_1 - 1) + T(n - q_2) + \Theta(n),$$

where $\Theta(n)$ is the total run-time for the new partition function. Note that, if we choose $q_1 = q_2 = q$, we obtain the same run-time recursion as the traditional quicksort.

In the case of threeway quicksort, the worst-case run time is given by

$$T_{worst}(n) = \max_{1 \leq q_1 \leq q_2 \leq n} \left[ T(q_1 - 1) + T(n - q_2) + \Theta(n) \right].$$

We prove that $T_{worst}(n) = O(n^2)$ using substitution method. In other words, we will prove that there exists three positive numbers $c_1$, $c_2$ and $N_0$ such that, for all $n \geq N_0$, we have

$$T_{worst}(n) \leq \max_{1 \leq q_1 \leq q_2 \leq n} \left[ c_1(q_1 - 1)^2 + c_1(n - q_2)^2 + c_2(n) \right]. \tag{1}$$

<u>Base Case</u> ($n = 1$): In this case, since there is only one entry in the array, we have $q_1 = q_2 = n = 1$. In other words,

$$T_{worst}(n = 1) \leq c_2.$$

The remainder of proof by induction is carried out, assuming that this condition holds true for all $n = 1, 2, \cdots$.

<u>Maintenance Case</u> ($n = k$): Assume that the inequality in Equation (1) is true for all $n = 1, \cdots, k$. Since $1 \leq q_1 \leq q_2 \leq k + 1$, we also have $q_1 - 1 \leq k$ and $k + 1 - q_2 \leq k$. Since the inequality in Equation (1) is true for $n = 1, \cdots, k$, we have $T(q_1 - 1) = O\left[(q_1 - 1)^2\right]$ and $T(k + 1 - q_2) = O\left[(k + 1 - q_2)^2\right]$. Then, the worst-case runtime for threeway quicksort for $n = k + 1$ reduces to

$$\begin{aligned} T_{worst}(k+1) &= \max_{1 \leq q_1 \leq q_2 \leq k+1} \left[ T(q_1 - 1) + T(k + 1 - q_2) + \Theta(k + 1) \right] \\ &\leq \max_{1 \leq q_1 \leq q_2 \leq k+1} \left[ c_1(q_1 - 1)^2 + c_2(k + 1 - q_2)^2 + c_3(k + 1) \right], \ \forall\, n \geq N_0, \end{aligned}$$

for some positive $N_0$.

Let us denote $f(q_1, q_2) = c_1(q_1 - 1)^2 + c_2(k + 1 - q_2)^2 + c_3(k + 1)$. Therefore, the gradient and Hessian of $f$ are given by

$$\nabla f = \begin{bmatrix} 2c_1(q_1 - 1) \\ 2c_2(q_2 - k - 1) \end{bmatrix}$$

and

$$\nabla^2 f = \begin{bmatrix} 2c_1 & 0 \\ 0 & 2c_2 \end{bmatrix}$$

respectively. Since the determinant of Hessian $\nabla^2 f$ is positive ($|\nabla^2 f| = 4c_1c_2 > 0$), the maximum always lies at the extreme points. This means either $q_1 = q_2 = 1$, or $q_1 = q_2 = k+1$. In both cases, we have

$$T_{worst}(k+1) \leq c \cdot k^2 + c_3(k+1), \ \forall\, n \geq N_0, \tag{2}$$

where

$$c = \begin{cases} c_2, & \text{if } q_1 = q_2 = 1, \\ \\ c_1, & \text{if } q_1 = q_2 = k+1. \end{cases}$$

Simplifying the RHS of the inequality in Equation (3), we have

$$\begin{aligned} T_{worst}(k+1) &\leq c\left[k^2 + 2k + 1\right] + c_3(k+1) - 2ck - c, \\ &\leq c(k+1)^2 + (c_3 - 2c)k + (c_3 - c), \end{aligned} \tag{3}$$

for all $n \geq N_0$. In other words, $T_{worst}(k+1) = O\left[(k+1)^2\right]$.

Termination Case: Assuming that this recursion terminates for some finite $n$, we have $T_{worst}(n) = O(n^2)$ by the principle of induction. □

## Problem 4. Sort by Frequency *25 points*

Write a program in Python that sorts all the integer entries in an input array $A$ of size $n$ according to the decreasing frequency of occurrence. If the frequency of two numbers is the same, then sort them in the increasing order of value. Assume that $A[j] \in \{0, 1, \cdots, k\}$ for all $j = 1, \cdots, n$, and let $k \ll n$ to allow enough number of repetitions.
(Hint: You can find frequencies using COUNTING-SORT).

**Example:** Let A = {3, 5, 2, 1, 0, 1, 2, 3, 4, 2, 0, 3, 4, 2, 1}. Note that $n = 15$ and $k = 5$. Let $f(i)$ denote the frequency of occurrence of a number $i$ in A. Then, we have

$$\begin{aligned} f(0) &= 2, & f(3) &= 3, \\ f(1) &= 3, & f(4) &= 2, \\ f(2) &= 4, & f(5) &= 1. \end{aligned}$$

Then, the output should look like: $B = \{2, 2, 2, 2, 1, 1, 1, 3, 3, 3, 0, 0, 4, 4, 5\}$.

**SOLUTION:** Code will be provided as a Jupyter notebook, along with these solutions. □

## Problem 5. Extra credit (5 points)

SELECTION-SORT$(A)$ sorts the input array $A$ by first finding the $j^{th}$ smallest element in $A$ and swapping it with the element in $A[j]$, in the order $j = 1, j = 2, \cdots, j = n-1$. Write pseudocode for SELECTION-SORT, and find the best-case and worst-case running times of SELECTION-SORT in $\Theta$-notation.

**SOLUTION:**

The pseudocode for SELECTION-SORT$(A)$ is given as follows:

```
SELECTION-SORT(A)
for i = 1 downto A.length
    j = MIN-INDEX(A, i)
    Swap A[i] with A[j]
```

```
MIN-INDEX(A, i)
k = i
for j = i + 1 downto A.length
    if A[j] < A[k]
        k = j
return k
```

If the size of array $A$ is $n$, then MIN-INDEX$(A, i)$ has a for-loop that runs for $\Theta(n)$ run-time in the worst case. Given that MIN-INDEX$(A, i)$ is inside the for-loop in SELECTION-SORT$(A)$ which itself iterates for $n$ times, it is natural that the runtime of SELECTION-SORT$(A)$ is $\Theta(n^2)$.

Note that the above discussion is an informal proof. A more formal way of proving the run-time for this algorithm is expected from students, which is similar to that presented for INSERTION-SORT$(A)$ algorithm in the class. $\square$

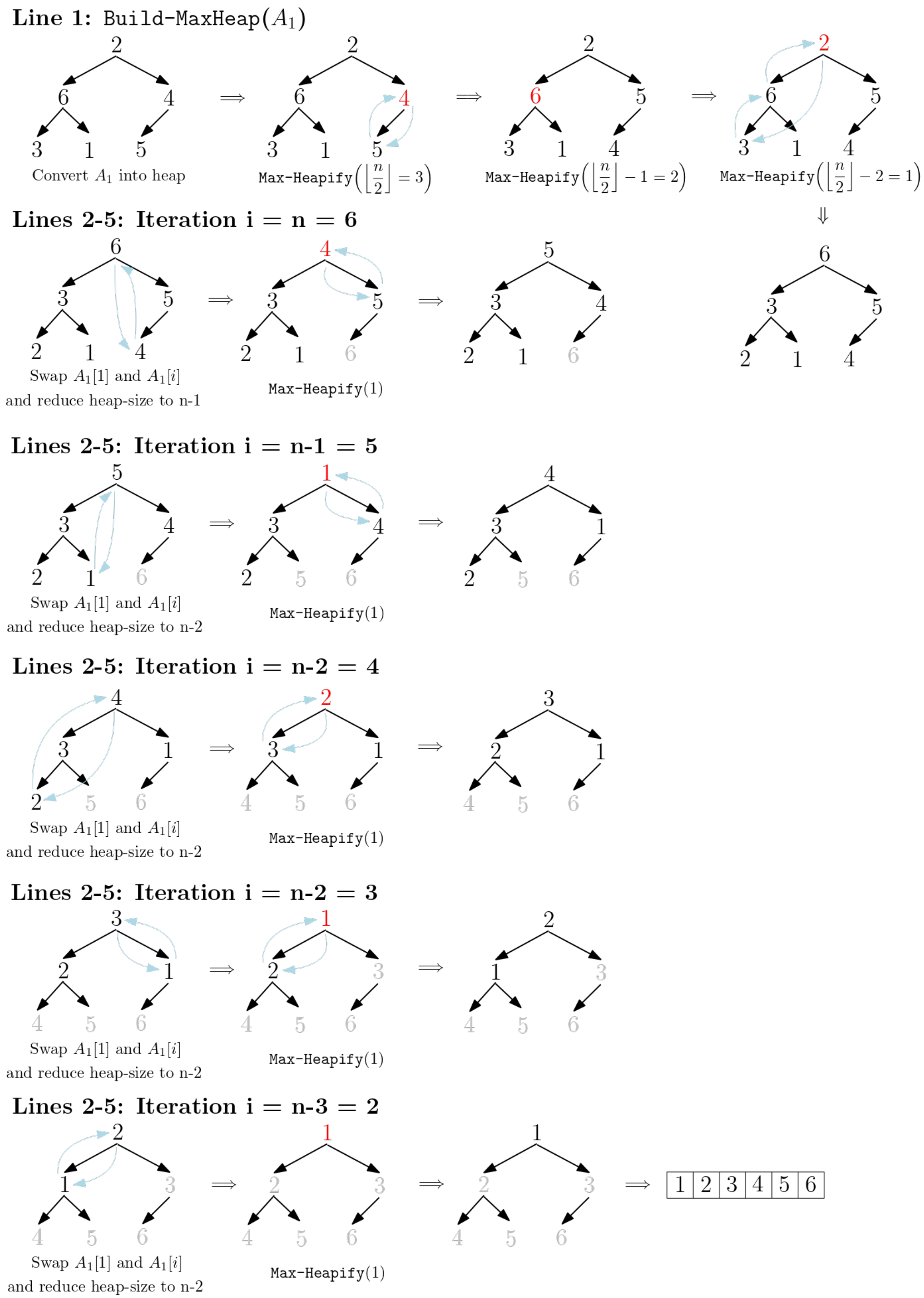


Figure 1: Stages of Heap-Sort on input $A_1 = \{2, 6, 4, 3, 1, 5\}$.

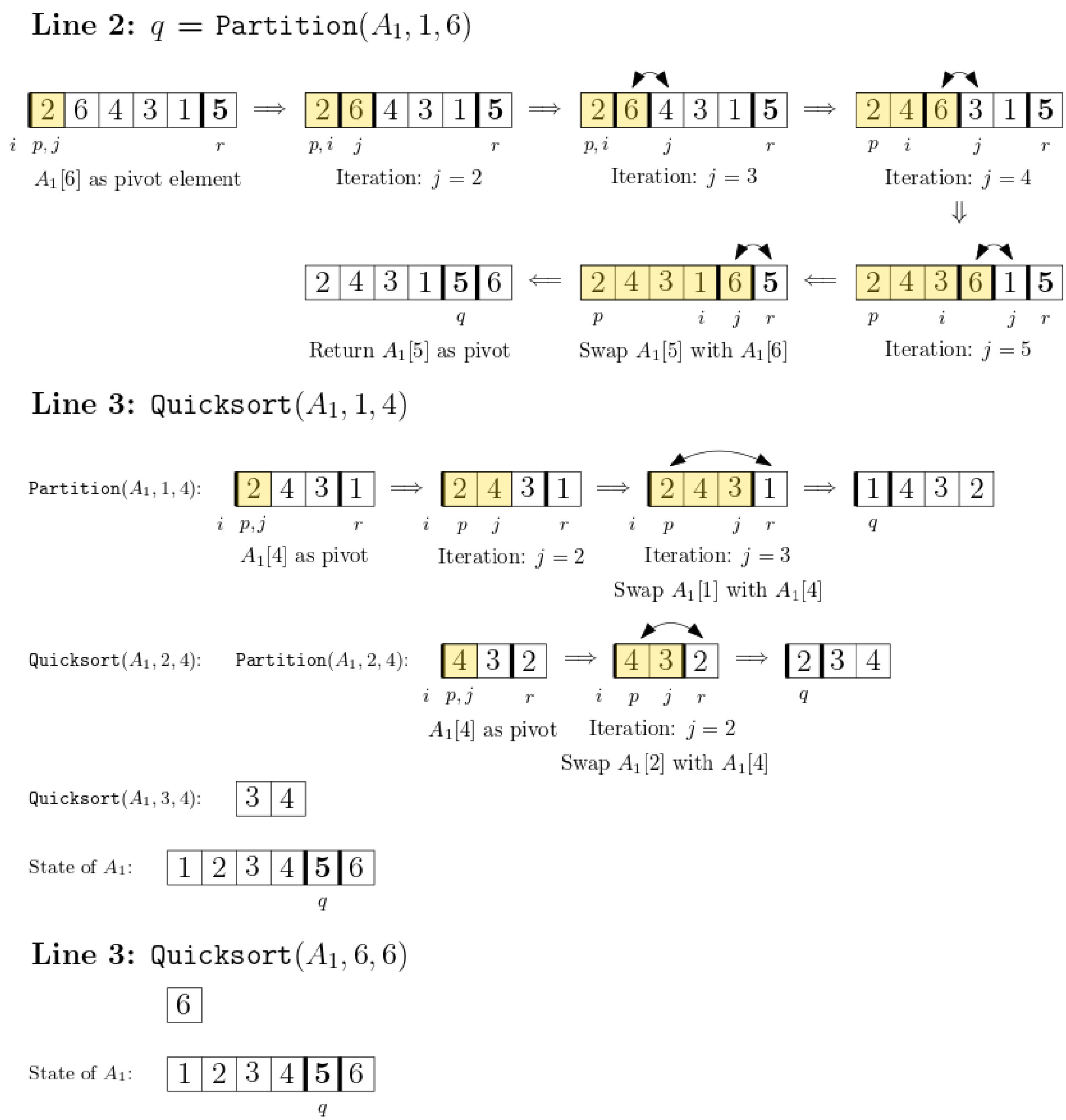


Figure 2: Stages of Quick-Sort on input $A_1 = \{2, 6, 4, 3, 1, 5\}$.