Daniel Raw

CSCI 303

Professor Liu

PA2 Write Up

My function random_selection operates by first choosing an element in the array randomly. Based on this pivot point, the for loops partition the array into 3 subarrays, but only undergoes recursion for the subarray with the kth element. As this process uses a for loop to iterate through the array and places the elements into a subarray so my first algorithm operates in O(n) time.

My function deterministic_selection operates by taking an array: arr and k: k_input as the parameters. I begin by calling a quicksort if the array has a length that is less than 10 elements. This is the simplest scenario and runs in O(n log(n)) time. However, if the array is larger than elements, the for loop divides the array into n/5 subarrays, that each contain 5 elements. The following for loop uses recursion to find the median of the median, and this section of my deterministic_selection function operates in T(n/5) time (worst case). The next portion of my deterministic_selection function partitions the array and either returns the calculated median of medians or using recursion, finds the median of the smaller or larger array. The worst case scenario, which has to undergo recursion, operates in O(n) time.

My quicksort_algorithm function operates by separating the array into 3 subarrays, and the subarrays are organized so that one holds the elements that are lower than the pivot point, equal to the pivot point, and higher than the pivot point, this runs in O(n) time. The function continues by using recursion on the subarrays that have values higher and lower than the pivot point. This is the best case scenario and operates in θ(n log(n)). The worst case scenario is when the pivot point is the smallest or largest element, and this runs in θ(n^2) time.

I generated a random list with n being 10^7 integers. I then called each function with list and made k = ⌈k/2⌉ and I stored how long it took to run. As expected, my random_selection was the fastest, my quicksort_algorithm being the second fastest, and my deterministic_selection being the slowest. Presented below are the outputs. When I made n = 10^8, the derministic_selection was a lot slower (proportionally) to random_selection and quicksort_algorithm. In conclusion, from this project, I learned that the first algorithm – random_selection, where a random pivot point is used to divide the array is the most efficient way. The random_selection algorithm performs best when the size is larger, as again, it chooses a random pivot point and hence, the split occurs near the middle of the list, making the sort faster.

```
In [1]: runfile('C:/Users/dsraw/Desktop/CSCI303/Project2/PA2.py', wdir='C:/Users/dsraw/Desktop/CSCI303/Project2')

What is your unordered array? [7, 2, 4, 6, 9, 11, 2, 6, 10, 6, 15, 6, 14, 2, 7, 5, 13, 9, 12, 15]

What is your k? 10
Generated a randomized array with a size if 10^7 in 16 seconds
Randomized selection of 50006 generated in 7 seconds
Deterministic selection of 50006 generated in 59 seconds
Quicksort of 50006 generated in 51 seconds
Randomized selection for a  given array:  7
Deterministic selection for a given array:  7
Quicksort for a given array  7

In [2]: runfile('C:/Users/dsraw/Desktop/CSCI303/Project2/PA2.py', wdir='C:/Users/dsraw/Desktop/CSCI303/Project2')

What is your unordered array? [7, 2, 4, 6, 9, 11, 2, 6, 10, 6, 15, 6, 14, 2, 7, 5, 13, 9, 12, 15]

What is your k? 10
Generated a randomized array with a size if 10^7 in 16 seconds
Randomized selection of 50017 generated in 7 seconds
Deterministic selection of 50017 generated in 61 seconds
Quicksort of 50017 generated in 52 seconds
Randomized selection for a  given array:  7
Deterministic selection for a given array:  7
Quicksort for a given array  7
```

```python
from random import randint, choice
from math import ceil
from time import time

start = False
while (start == False):
    question = input("What is your unordered array? ")
    A = list()
    question = question.replace("[","")
    question = question.replace("]","")

    for i in question.split(","):
        A.append(int(i)) #raise an error if the number is not an integer
    if (len(A) > 1): #check for an array that has at least 2 elements
        start = True

k = int(input("What is your k? "))
while (k < 1 or k > len(A)):
    k = int(input("What is your k? ")) #check to see if a proper k is inputted

#Algorithm 1 (10 points)
def random_selection(arr, k_input):
    #used the algorithm described in the textbook

    split = choice(arr) #from random import, get the random pivot
    #partitoning the subarrays
    below = list()
    high = list()
    equal = list()
    for i in arr:
        if (i > split): #when the iteration is larger than the split
            high.append(i)
        elif (i < split): #when the iteration is smaller than the slit
            below.append(i)
        elif (i == split): #when the iteration is equal to the split
            equal.append(i)

    if (len(below) >= k_input):#recursive call
        return random_selection(below, k_input)
    elif(len(below) == k_input - 1):#if it is the median, return the split
        return split
    elif (len(below) < k_input - 1):#if the length of the below and equal list is greater than the k, return split
        if (len(equal)+len(below) >= k_input):
            return split
        return random_selection(high, k_input - len(below) - len(equal)) #otherwise call the recursive call
```

```python
#Algorithm 2 (20 points)
def deterministic_selection(arr, k_input):
    if(len(arr) <= 10): #sort when the length is less than or equal to 10, call quicksort
        return quicksort(arr, k_input)
    #dividing the A into n/5 subarrays
    tracker = []
    array = []
    for i in range(len(arr)):
        if (len(array) == 5): #corner case, when the length of the array is 5, call add_partition
            tracker = add_partition(tracker, array)
            array = []
        elif (len(array) < 5 and i != len(arr)-1): #append to array
            array.append(arr[i])
        elif (i == len(arr)-1): #append to array
            array.append(arr[i])
            if (len(array) != 0): #in corner case scenerios, call add_partition
                tracker = add_partition(tracker, array)
    index = 0
    #finding the median of the median using recursion
    for i in tracker: #recursive call scenerios
        if (len(i) == 5):
            tracker[index] = deterministic_selection(i,3)
        else:
            tracker[index] = deterministic_selection(i, ceil(len(i)/2)) #divide by 2
        index = index + 1
    recurs = deterministic_selection(tracker, ceil(len(arr)/10)) #divide length S by 10
    #using the median to partition the array
    below = []
    high = []
    equal = []
    for i in arr: #similar recursive loop as in my first algorithm
        if (i > recurs):
            high.append(i)
        elif (i < recurs):
            below.append(i)
        elif (i == recurs):
            equal.append(i)
    #using recursion on the smalller or larger subarray
    if (k_input <= len(below)):
        return deterministic_selection(below, k_input)
    elif (k_input > len(below) + len(equal)):
        return deterministic_selection(high, k_input - len(below) - len(equal))
    #or simply returning the median
    else:
```

```python
        return recurs

def add_partition(arr, array):#function that helps me append the partitions
    arr.append(array)
    return arr

#Algorithm 3 (20 points)
def quicksort(arr, k):
    q = quicksort_algorithm(arr)
    return q[k-1] #since the index starts at 0 we have to index to k-1 for the kth smallest in a sorted array

def quicksort_algorithm(arr):
    below = list()
    high = list()
    equal = list()
    if (len(arr) <= 3): #if there are only 3 elements, simply sort. This is the base case
        return sorted(arr)
    else:#recursive case
        split = choice(arr)#random pivot
        for i in arr:#iterate through S
            if (i < split):
                below.append(i)
            elif (i > split):
                high.append(i)
            elif (i == split):
                equal.append(i)
        return quicksort_algorithm(below) + equal + quicksort_algorithm(high) #recursion for both subarrays

def main():
    A = list()
    count = 0
    start_array = time()

    while (count != 10000000):
        A.append(randint(0,100000))
        count = count + 1
    end_array = time()
    print("Generated a randomized array with a size if 10^7 in {} seconds ".format(ceil(end_array - start_array)))
    start_randSelect = time()
    selection_print = random_selection(A, ceil(count/2))
    end_randSelect = time()
    print("Randomized selection of {} generated in {} seconds".format(selection_print, ceil(end_randSelect - start_randSelect)))
    start_determ = time()
```

```
│   ├── deterministic_print = deterministic_selection(A, ceil(count/2))
│   ├── end_determ = time()
│   ├── print("Deterministic selection of {} generated in {} 
seconds".format(deterministic_print, ceil(end_determ - start_determ)))
│   ├── start_quick = time()
│   ├── quicksort_print = quicksort(A, ceil(count/2))
│   ├── end_quick = time()
│   └── print("Quicksort of {} generated in {} seconds".format(quicksort_print, 
ceil(end_quick - start_quick)))
├── main()
├── print("Randomized selection for a  given array: ", random_selection(A, k))
├── print("Deterministic selection for a given array: ", deterministic_selection(A, 
k))
└── print("Quicksort for a given array ", quicksort(A, k))
```