# Midterm Exam

CIS-675 DESIGN AND ANALYSIS OF ALGORITHMS

PROF. IMANI PALMER

4/28/2021

Anthony Redamonti
SYRACUSE UNIVERSITY

Questions 3-6 have been answered.

## Question 3:

You are given an array $A$ of distinct integers, sorted in decreasing order. We want to determine if there is any index $i$ such that $A[i] = i$ (assume the indexing starts at 0). Design an efficient divide-and-conquer algorithm to solve this. What is its running time? Explain your answer.

Example 1: $N = 6, \ A = [100, 50, 20, 10, 4, -1]$

$\boldsymbol{A[0] = 100}, \boldsymbol{A[1] = 50}, \boldsymbol{A[2] = 20}, A[3] = 10, A[4] = 4, A[5] = -1$

No point in searching indexes less than mid since the values are getting larger and the indices are getting smaller.

Example 2: $N = 6, \ A = [5, 1, 0, -1, -2, -3]$

$A[0] = 5, A[1] = 1, \boldsymbol{A[2] = 0}, \boldsymbol{A[3] = -1}, \boldsymbol{A[4] = -2}, \boldsymbol{A[5] = -3}$

No point in searching indexes greater than mid since the values are getting smaller and the indices are getting larger.

To solve this problem efficiently, use a binary search algorithm. Using the rules outlined in the problem statement, the input array can be recursively divided by eliminating sections that could not possibly have the desired index.

```
#include <stdio.h>

// recursively split the input into 2 halves until either answer found or not fou
nd.
int binary_search(int* array, int left, int right){

   int range = right-left;
   int midpoint = left + (range/2);

   if(array[midpoint] == midpoint){ // if an answer is found, return it.
      return midpoint;
   }

   // If no answer has been found, return -1.
   // If the array cannot be split anymore, return.
   if(midpoint == left){
      return -1;
   }

   // if the index is greater than the value, do not look beyond the index.
   if(array[midpoint] < midpoint){
```

```
      return(binary_search(array, left, midpoint));
   }

   // if the index is less than the value, do not look before the index.
   if(array[midpoint] > midpoint){
      return(binary_search(array, midpoint, right));
   }
}

int main(){
   int i;
   int array[7] = {12, 7, 6, 3, -2, -5, -11};
   int n = 7;
   printf("original array: [");
   for(i = 0; i < n-1; i++){
      printf("%d, ", array[i]);
   }
   printf("%d]\n", array[n-1]);
   int index = binary_search(array, 0, n);
   printf("index: %d\n", index);
}
```

If the index at the midpoint of the array is greater than the value held in the array at that index, the search does not look at the second half of the array. The reason is that it is impossible for the value to match the index as the indices are increasing and the values are decreasing. Likewise, if the index at the midpoint is less than the value, it is impossible for indices before the midpoint to match their values as the values would also be greater than the respective indices. In this case, the search does not look at the first half of the array.

The output of the code is below.

```
original array: [12, 7, 6, 3, -2, -5, -11]
index: 3
```

The input is continually divided into two halves.

The recursive loop will run until $\frac{N}{2^k} \leq 1$, where $N$ is the size of the input.

This simplifies to: $N = 2^k$.

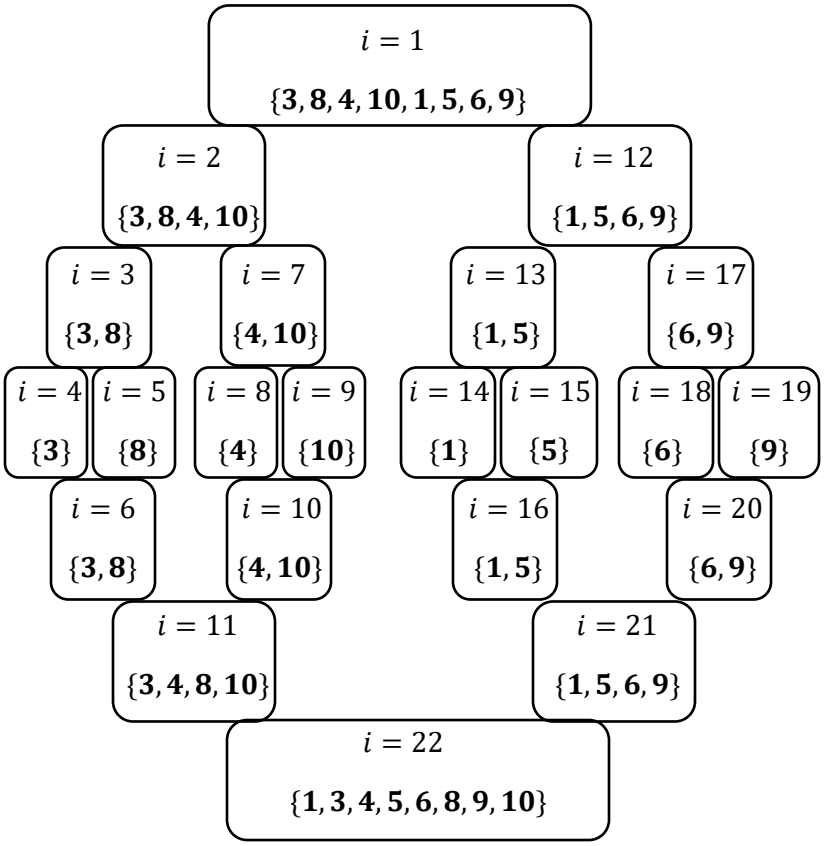
Applying $\log_2$ to both sides: $k = \log_2 N$

The runtime of this algorithm is $O(\log_2(N))$, which simplifies to $O(\log(N))$.

## Question 4:

Given the following array as input, illustrate how the Mergesort algorithm performs. To illustrate the Mergesort's behavior, start with the dividing of the array until the end condition of the recursive function is met and then show how the merge is performed.

$Array$: $\{3, 8, 4, 10, 1, 5, 6, 9\}$



## Question 5:

Buzz Sort is a sorting algorithm that goes like this:

If the sequence length is at most 4, sort is using bubble sort.

Else:

1. Divide the list into 5 pieces evenly by scanning the entire list.
2. (recursively) sort the first 3/5 of the list.
3. (recursively) sort the last 3/5 of the list.
4. (recursively) sort the first 3/5 of the list.

For example, on the input sequence: 1,5,3,2,4

The first recursive sort produces: 1,3,5,2,4

The second sort produces: 1,3,2,4,5

The third sort produces: 1,2,3,4,5

Give an example sequence on 5 or 10 integers where Buzz Sort does not terminate with the correct answer.

The following 5 integer input sequence will yield an incorrect answer using Buzz Sort.

8, 7, 6, 5, 4

The first recursive sort produces: **6, 7, 8**, 5, 4

The second sort produces: 6, 7, **4, 5, 8**

The third sort produces: **4, 6, 7**, 5, 8

Thus, the answer provided by Buzz Sort is incorrect.

In a list of 5 elements, Buzz Sort has one index that is shared between the first and last three-fifths of the list. The index is index 2 shown below.

$$\{0, 1, 2, 3, 4\}$$

Therefore, it can only move one element per recursive sort between each group. If elements in indices 0 and 1 need to be moved to indices 3 and 4 and elements in indices 3 and 4 need to be moved to indices 0 and 1, Buzz Sort does not perform enough sorting operations on each group to complete the sort of the list. Hence, any list of length 5 sorted in reverse order will cause Buzz Sort to fail.

## Question 6:

Give a careful, step-by-step, proof for the following statement. If $n^2 - 1$ is even, then $n$ is odd.

Claim: If $n^2 - 1$ is even, then $n$ is odd.

Proof: Direct Proof: Suppose $n^2 - 1$ is even. Let $a$ be an even integer such that $a = 2m$, where $m$ is an integer. Therefore, the following must be true: $n^2 - 1 = 2m$.

Solving for $n$:

$n^2 = 2m + 1$

$n = \sqrt{2m + 1}$

The expression $2m$ will always produce an even number by the multiplicative property of even and odd integers. The property states that any even number multiplied by an odd or even number will always produce an even result.

Hence, $2m + 1$ will always produce an odd result as the sum of any even number and 1 will always be odd by the additive rule of even and odd integers.

Finally, the square root of an odd number must also be odd.

This is the same as the following expression: $\sqrt{2m + 1} = 2m + 1$.

Squaring both sides gives: $2m + 1 = (2m + 1)^2 = 4m^2 + 4m + 1$.

As stated earlier, $4m^2$ and $4m$ will always produce even numbers based on the multiplicative property of even and odd numbers.

$4m^2 + 4m$ will always produce an even number, due to the additive property of even and odd numbers which states that the sum of two even numbers is also even.

As stated earlier, the sum of any even number and 1 will always be odd by the additive rule of even and odd integers. Therefore, the result of $4m^2 + 4m + 1$ will always be odd. Hence, the square root of an odd number must also be odd.

Therefore, $\sqrt{2m + 1}$ must produce an odd integer, and since $n = \sqrt{2m + 1}$, $n$ must be an odd integer.

□