# Graded Assignment 2 – DSA

Claudio Maggioni

March 24, 2019

## Contents



## Listings

# 1 Exercise 2

## 1.1 Exercise a

The pseudocode for *Sum of two* can be found in listing 1. The total cost of this algorithm in the worst case is the sum of the worst case of mergesort ($O(nlog(n))$) and the cost of the worst case in the partition done afterwards (which is equivalent to not finding the sum, i.e. $2n = O(n)$). Therefore, the total cost is $\theta(nlog(n))$.

```
FUNCTION SUM-OF-TWO(A, s):
  A ← mergesort(A)
  i ← 1
  j ← A.length

  while i < j:
    sum ← A_i + A_j
    if sum = s:
      return TRUE
    elif sum > s:
      j ← j - 1
    else:
      i ← i + 1

 return FALSE
```

Listing 1: Sum of two in pseudocode

## 1.2 Exercise b

The pseudocode for *Sum of three* can be found in listing 2. SEARCH-TWO has a time cost of $O(n)$ in the worst case (if no elements are found), and the loop of SEARCH has an added cost of $O(n)$. The total cost in the worst case then, including mergesort, is $n^2 + nlog(n) = \theta(n^2)$.

```
FUNCTION SEARCH-TWO(A, sum2, i_skip):
  i ← 1
  j ← A.length

  while i < j:
    if i = i_skip:
```

```
      i ← i + 1
    elif j = i_skip:
      j ← j - 1
    else:
      sum ← A_i + A_j
      if sum = sum2:
        return TRUE
      elif sum > sum2:
        j ← j - 1
      else:
        i ← i + 1

  return FALSE

FUNCTION SUM-OF-THREE(A, s):
  A ← mergesort(A)
  l ← A.length

  for i from 1 to l:
    if SEARCH-TWO(A, s - A_i, i):
      return TRUE

  return FALSE
```

Listing 2: Sum of three in pseudocode

## 1.3 Exercise c

The *Python* code used to implement *Sum of three* can be found in the listing 3.

```
#!/usr/bin/env python3

import sys

def search_two(A, sum2, i_skip):
    i = 0
    j = len(A) - 1

    while i < j:
        if i == i_skip:
            i = i + 1
        elif j == i_skip:
```

```
            j = j - 1
        else:
            cs = A[i] + A[j]
            if cs == sum2:
                return True
            elif cs > sum2:
                j = j - 1
            else:
                i = i + 1

    return False

def sum_of_three(A, sum3):
    A.sort() # assume using mergesort for worst case of O(n*log(n))
    l = len(A)

    for i in range(l):
        if search_two(A, sum3 - A[i], i):
            return True

    return False

if __name__ == "__main__":
    args = [int(x) for x in sys.argv[1:]]
    print(sum_of_three(args[1:], args[0]))
```

Listing 3: Sum of three in Python