

Assignment 3

Marcel Visser Student number: 0804065
marcel1337@hotmail.com

March 10, 2013

Exercise 1

no idea

Exercise 2

a

```

function DUPEFIND(array  $A$ )

  //  $A$  is an array of length  $n$ 
  //  $-1$  is taken as a value it returns once it did not find duplicates

   $n = A.length$ 
  for  $i = 0 \leq A.length$  do
    if  $A[i] = A[i-1]$  then
      return  $i$ 
    else if  $i = A.length$  then
      return  $-1$ 
    end if
  end for
end function

```

Loop invariant: At the start of each iteration, the array $[0..i-1]$ does not contain any duplicates.

- *Initialization:*
Before the first iteration, $i = 0 \Rightarrow A[0..-1]$ ($A[empty]$) does trivially not contain any duplicates.
- *Maintenance:*
During every iteration, i increases and $[0..i-1]$ will still not contain any duplicates.
- *Termination:*
-The for loop either ends as soon as $i=n+1 \Rightarrow n = i-1$ which implies $A[0..i-1] = A[0..n]$, so that there are no duplicates found whatsoever. It will return a NIL value in this case.
-**Or** it ends as soon as a duplicate is found with $A[i]=A[i-1]$, which implies $A[0..i-1]$ does contain a duplicate. $A[i..n]$ however may still have another duplicate somewhere in the sorted array. The algorithm will return $A[i]$.

So this means the algorithm will have at most n iterations (or less if a duplicate is found), which gives it a running time of $O(n)$.

The minimum operations could be 1, so the lower bound would be $\theta(1)$.

b

Assume an algorithm with $2n-2$ comparisons. If this algorithm receives array $A[5]$ and array $B[6]$, this algorithm will only need one comparison to merge these two (checking if $A < B$ or vice versa. So $2n-1$ ($2 * 1 - 1$) comparisons are sometimes necessary.

Exercise 3

a

```
function MINMAX(array  $A$ )  
  
  //  $S$  is an array of length  $n$   
  
   $n = A.length$ ;  
  while  $n > 2$  do Remove( $S$ , Oracle( $S[n-3]$ ,  $S[n-2]$ ,  $S[n-1]$ ));  $n = S.length$ ;  
  end while  
  return  $S$ ; //equivalent to  $X$  now  
end function
```

Loop invariant: At the start of each iteration, $S[n-3, n-2, n-1]$ contains at least one value which is not the minimum or maximum.

- *Initialization:*
Before the first iteration, $S[n-3, n-2, n-1]$ has a median $\Rightarrow S[n-3, n-2, n-1]$ has a value which is not the minimum or maximum.
- *Maintenance:*
During every iteration, $S[n-3, n-2, n-1]$ will still contain at least one value which is not the minimum or maximum. This value will be removed. Then n decreases.
- *Termination:*
Once $n \leq 2$, $S[n-3, n-2, n-1]$ becomes $S[n-2, n-1]$. All medians (value which is not the minimum or maximum) will have been removed. There is no longer a median present in this new array, so the new array contains the minimum and the maximum.

Running time? Dependant on the length of the array, linearly so $\theta(n)$.

b

```
function SORTORACLE(array A)
```

```
// A is an array of length n
```

```
n = S.length; X = minMax(S);
```

```
  if X[0] > X[1] then max = X[0]; min = X[1];
```

```
  else max = X[1]; min = X[0];
```

```
  end if mergeSort(); with comparisons switched. Oracle(min,a,b) = return smallest be it  
  a or b. Oracle(a,b,max) = return largest be it a or b.
```

```
end function
```

The validity of mergeSort has already been proven. This algorithm works as it only modifies the base mechanics of mergeSort and thus does not influence the working of it. It switches the comparisons with instant returns of the requested value. The running time of mergeSort is $O(n \log n)$, where as of *minMax* is $O(n)$. So the total running time is $O(n \log n) + O(n) = O(n \log n)$.

Exercise 4:

A:

0			
1	20		
2			
3			
4	16	5	
5	44	88	11
6	94	39	
7	12	23	
8			
9	13		
10			

B:

0	11
1	39
2	20
3	5
4	16
5	44
6	88
7	12
8	23
9	13
10	94

C:

0	11
1	23
2	20
3	16
4	39
5	44
6	94
7	12
8	88
9	13
10	5

- b) For all keys the amount of probes was 0;
 except for the following list denoted as <key> :< amount >
88:1 23:1 94:4 11:6 39:7 20:1 5:10
- c) For all keys the amount of probes was 0;
 except for the following list denoted as <key> :< amount >
88:1 23:1 11:2 39:3 20:2 16:2 5:3