

CMPE 300 ANALYSIS OF ALGORITHMS
MIDTERM ANSWERS

1.

- a) This fragment contains just two operations, independent of the data size. So the time complexity is constant, $\Theta(1)$.
- b) The outer loop is executed a constant number of times (3 times) and the inner loop is executed n times. So, the time complexity is $3n \in \Theta(n)$.
- c) The loop is executed n^2 times, so the complexity is $\Theta(n^2)$.
- d) The outer loop is executed n times. Each execution consists of $(n+n*\log n)$ operations. So, the complexity is $n*(n+n*\log n) \in \Theta(n^2 \log n)$.
- e) The outer loop is executed n times. For each iteration i , the number of executions of the inner loop depends on where the number i is placed in the array. Since the array contains a permutation of the numbers from 1 to n , we know that each number i will occur in the array once. That is, the inner loop will be executed 1 time for some value of i , 2 times for some value of i , ..., and n times for some value of i . So, the total number of operations will be $\sum_{i=1}^n i = \frac{n(n+1)}{2} \in \Theta(n^2)$.
- f) Corresponding to the iterations of the outer loop, the number of operations inside the loop is 1, n , 1, n , 1, n , ..., 1, n . Here, there are $n/2$ 1's and $n/2$ n 's. So, the complexity is $(n/2)*1 + (n/2)*n \in \Theta(n^2)$.

2. (See the Assignment #1 answers.)

3.

- a)
$$T(n) = T\left(\frac{9}{10}n\right) + n = T\left(\frac{9^2}{10^2}n\right) + \frac{9}{10}n + n = T\left(\frac{9^3}{10^3}n\right) + \frac{9^2}{10^2}n + \frac{9}{10}n + n = \dots$$
$$= T(1) + n \sum_{i=0}^{k-1} \left(\frac{9}{10}\right)^i = n \frac{(9/10)^k - 1}{(9/10) - 1} = 10n \left(1 - \frac{9}{10}^{\log_{10/9} n}\right) \in \Theta(n), \text{ since } \frac{9}{10}^{\log_{10/9} n} < 1$$
- b) According to Master Theorem in the lecture notes, $a=2$, $b=4$, $d=0.5$. Since $a=b^d$, $T(n) \in \Theta(n^d \log n) = \Theta(\sqrt{n} \log n)$
- c) $T(n) = T(\sqrt{n}) + 1 = T(\sqrt[2]{n}) + 2 = T(\sqrt[3]{n}) + 3 = \dots = T(\sqrt[i]{n}) + i$
We try to make $T(\sqrt[i]{n}) = T(2)$. That is, $n^{1/2^i} = 2$. So, $n = 2^{2^i}$. Thus, $i = \log \log n$.
So, $T(n) = T(2) + \log \log n \in \Theta(\log \log n)$.

4.

```
function BinarySearch (L[1:∞], X)           // The representation of the array
    low=1                                   // (L[1:∞]) can be any representation.
    high=1
    while (L[high]<X) do
        low=high+1
        high=high*2
    endwhile
    // Now we have bounded X between L[low] and L[high]. We will execute the
    // original binary search routine.
    found=false
```

```

while (not found) and (low≤high) do
  mid=floor((low+high)/2)
  if X=L[mid] then
    found=true
  else
    if X<L[mid] then
      high=mid-1
    else
      low=mid+1
    endif
  endif
endif
endwhile
if found then
  return(mid)
else
  return(0)
endif
end

```

The explanation of the algorithm is as follows: Initially, we do not know the position n in the array that holds X , nor do we know the size of the array (which can be arbitrarily larger than n). What we do is to begin at the left side, and start searching for X . The secret is to jump twice as far to the right at each iteration. Thus, we will initially search array positions 1, 2, 4, 8, 16, 32 and so on. Once we have found a value that is larger than or equal to what we are searching for, we have bounded the subrange of the array from our last two searches. Then we perform original binary search within this range.

The length of the subarray we determined is at most n units wide, and we have found this subarray in at most $\log n+1$ steps. The original binary search of the subarray will find the position n in an additional $\log n$ operations at most, for a total cost in $O(\log n)$ operations.