21st CSEC – Past Year Paper Solution (2016 – 2017 Semester 1)
CZ/CE2001 - Algorithms

1 **(a)** **(i)** TRUE

(ii) FALSE

(iii) TRUE

(iv) FALSE

(v) FALSE

(b) TRUE

Since $f(n) \in \Omega(h(n))$, we must have $\lim_{n\to\infty} \frac{f(n)}{h(n)} = c$ for some $0 < c < \infty$

Similarly, we must have $\lim_{n\to\infty} \frac{g(n)}{h(n)} = d$ for some $0 < d < \infty$

By the limit law, we multiply both equations to get $\lim_{n\to\infty} \frac{f(n)g(n)}{h(n)^2} = cd$

Hence, since we must have $0 < cd < \infty$, so we conclude that $f(n)g(n) \in \Omega(h(n))$

(c) **(i)** The algorithms will compute the largest element on array

(ii) Suppose that time cost if there are n elements on the array is T(n)
We have that T (1) = 0 and T(n) = 1 + T(n-1)
Solving this:
T(n) = T(n-1) + 1 = T(n-2) + 1 + 1 = T(n-2) + 2 = … = T (1) + n-1 = n-1
So, we can approximate the time complexity is about O(n)

(d) **(i)**
- Best case is when the first element is equal to second element. In this case there will be just 1 comparison
- Worst case is if all the elements are different. Hence, we do all the comparison
  The number of comparisons is $(n-1) + (n-2) + \cdots + 1 = \frac{n(n-1)}{2} \sim O(n^2)$

(ii) The list of all possibilities with number of comparisons:

Array	# Comparison
[a,a,a]-	1
[a,a,b]	1
[a,b,a]	2
[a,b,b]	3
[b,a,a]	3
[b,a,b]	2
[b,b,a]	1
[b,b,b]	1

Hence, the average-case number of comparisons are:

$$\frac{1}{8}(1+1+2+3+3+2+1+1) = \frac{1}{8}(14) = \frac{7}{4} = 1.75$$

If there are errors, please report using the form in bit.ly/SCSEPYPError

2 **(a)** **(i)** We will calculate the value of $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ to determine the result

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n\ \log n}{n\ \log n^2 + n^{1.1}} = \lim_{n\to\infty} \frac{n\ \log n}{2n\ \log n + n^{1.1}}$$

$$= \lim_{n\to\infty} \frac{1}{2 + \frac{n^{0.1}}{\log n}}$$

We know that $\lim_{n\to\infty} \frac{n^{0.1}}{\log n} = \infty$, hence we have the limit is equal to 0

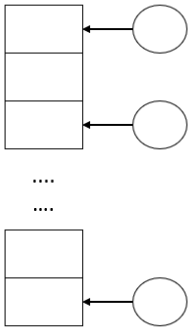
So, we have $f(n)\ \epsilon\ O(g(n))$ only

(ii) Same as (i), we calculate the limit

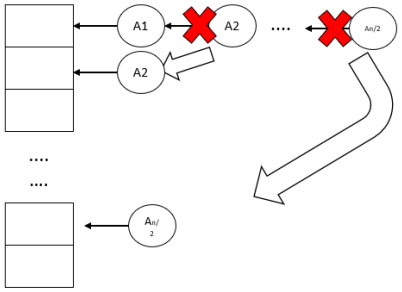
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n\,!}{2^n} = \infty$$

Hence, we have $f(n)\ \epsilon\ \Omega(g(n))$ only

(b) **(i)** **Best scenario**: the $\frac{n}{2}$ keys are hashed and distributed evenly into the n slots and no rehashing.



Worst scenario: the $\frac{n}{2}$ keys are hashed in consecutive slots in the table. Each key always must rehash and visit every key in the table. The ith key is hashed and rehashed i times to get the slot.



(ii) **Best case:** Assuming that equal probability for a key to be hashed into each of the n slots, the average-case time complexity:

$$\frac{1}{n}\sum_{i=1}^{\frac{n}{2}} 1 = \frac{1}{n}.\frac{n}{2} = 0.5 \sim O(1)$$

If there are errors, please report using the form in bit.ly/SCSEPYPError

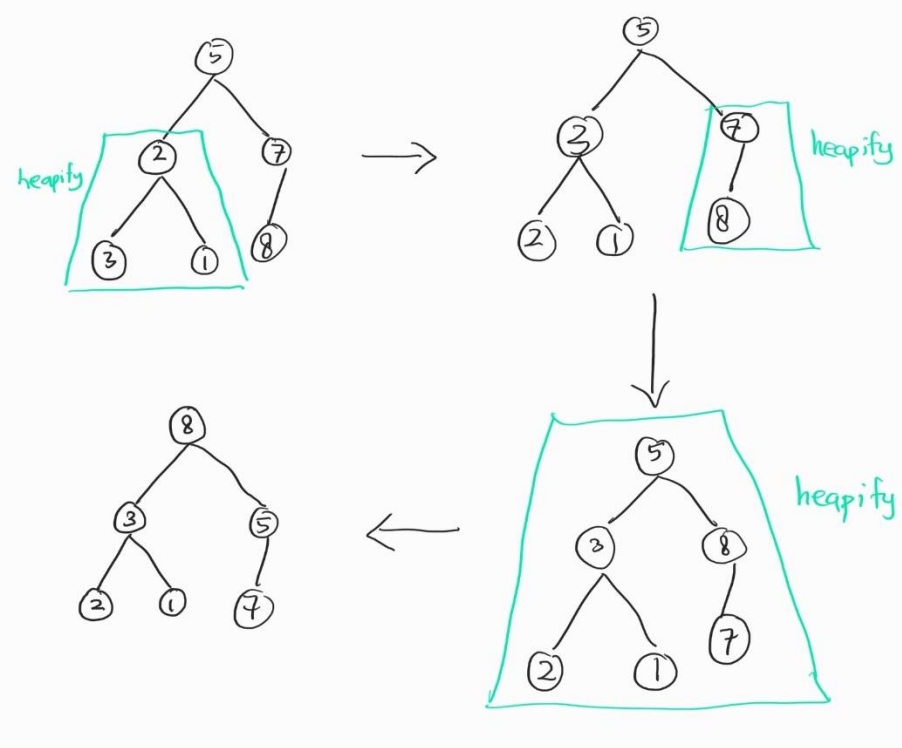
Worst case: Assuming that equal probability for a key to be hashed into each of the n slots, the average-case time complexity:

$$\frac{1}{n}\sum_{i=1}^{\frac{n}{2}} i = \frac{1}{n}.\frac{\frac{n}{2}\left(\frac{n}{2}-1\right)}{2} = \frac{n-2}{8} \sim O(n)$$

(c) **(i)** Optimization Problem: Find minimal number of vertices in the vertex cover
Time Complexity Class: NP-hard

(ii) Decision Problem: Given graph G and positive integer k, does G have vertex cover size at most k?
Time Complexity Class: NP-complete

3 **(a)**



Number of comparisons = 3 + 2 + 3 = 8

(b) Suppose we have array of n elements
We have that for first step, the left side will be empty, and the right will remain the same.
Hence, for first step, number of comparisons needed is (n-1).
For the next step, we have that the remaining (n-1) elements is also in ascending order and so, the number of comparisons will be (n-2).
Going through until the last one, we will have the total number of comparisons is

$$(n-1) + (n-2) + \cdots + 1 = \frac{n(n-1)}{2} \sim O(n^2)$$

If there are errors, please report using the form in bit.ly/SCSEPYPError

(c)

```
def negativeKeys(array A){
   first_pos = 0
   for(i=0;i<n;i++){
       if(A[i] < 0){
         swap(A[i],A[first_pos])
         first_pos += 1
       }
   }
}
```

Since in each iterative step, the only comparison is only A[i] with 0, so we have constant-time complexity for one iterative step. Because there are n steps, hence time complexity is approximately O(n)

(d) First, we construct the Heap using the algorithm constructHeap(H), which have worst case time complexity is O(n)

Next, we will use deleteMax(H) function k times to extract the k largest value at each time the function is called. Hence in this step, worst time complexity is O(k log n) because at each step, the worst time complexity is O(log n) and we do it k times.

Put it together, we have worst time complexity as O(n + k log n)

4 **(a)** **(i)**

1	→ 2→ 4
2	→ 1→ 3
3	→ 2
4	→ 1

(ii)

```
def convertAdjacencyMatrix(array A){
   for(i=0;i<n;i++){
       for(j=i;j<n;j++){
           if (A[i][j] == 1)
              add j to linked list with key i
       }
   }
}
```

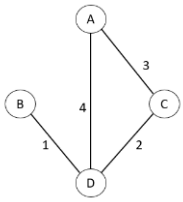
Time complexity can be calculated by find number of iterative steps, as we only do constant time calculation in each step.

Number of iterative steps are:

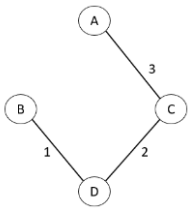
$$n + (n-1) + (n-2) + \cdots + 1 = \frac{n(n+1)}{2} \sim O(n^2)$$

If there are errors, please report using the form in bit.ly/SCSEPYPError

(b) No, there are counter example as below:

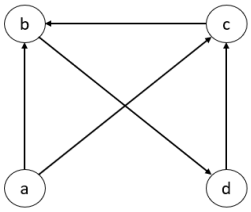


In this graph, the MST can be find using Prim's Algorithm, and resulted as below:



Hence in this MST, the shortest path from A to D is via C, which is A-C-D, the weight is 5. However, in the original path, the shortest path from A to D is direct A-D, the weight is 4. So, the path between two vertices in MST is not always the shortest path between two vertices in graph

(c) **(i)**



(ii) We will create algorithm by observing the middle vertices. That is for each vertices v, we will find whether exist vertices u and w such that there is edge u→v and v→w.

```
def findSquareGraph(array A){
   for(i=0;i<n;i++){
       for(j=0;j<n;j++){
           for(k=0;k<n;k++){
              if (G[j][i] == 1 && G[i][k] == 1)
                 Sq[j][k] = 1
           }
       }
   }
}
```

Time complexity is $O(n^3)$ because there are n^3 iterative steps (we check every combination of two vertices for each vertices) and in each step it takes constant time complexity

Solver: Andrew Wiraatmaja (andr0079@e.ntu.edu.sg)