2021학년도 2학기 자료구조 중간고사

2021. 10. 21

점수	

학번: **성명:**

※ (1 - 20) 아래 문제를 읽고 물음에 답하시오. [100점 각 5점]

1. 시간 복잡도 함수에 대한 점근 표기법으로 옳은 것만을 모두 고르면?

> ㄱ. $n^{2n} + 6 \cdot 2^n = \boldsymbol{O}(n^{2n})$
> ㄴ. $n^2/\log n = \boldsymbol{\theta}(n^2)$
> ㄷ. $n^3 2^n + 6n^2 3^n = \boldsymbol{O}(n^2 2^n)$
> ㄹ. $6n^3 + \log n = \boldsymbol{O}(n^3)$

① ㄱ, ㄷ　　**② ㄱ, ㄹ**
③ ㄴ, ㄷ　　④ ㄴ, ㄹ

2. 다음 파이썬 함수의 시간 복잡도를 빅오(O) 표기법으로 표현한 것은? (단, n > 1)

```
def testing(n) :
    sum = 0
    for i in range(n) :
        j = n
        while( j > 1 ):
            sum += 1
            j //= 2
```

① O(n)
② $O(n^2)$
③ O(nlogn)
④ O(logn)

3. 일반 리스트(general list) $L = (e_1, e_2, \ldots, e_n)$에 대해, 함수 head($L$)은 첫 번째 원소인 e_1을 반환하고, 함수 tail(L)은 리스트 $(e_2, e_3, \ldots, e_n)$를 반환한다고 할 때, 다음과 같은 리스트 A에 대한 함수의 수행 결과 중 문자 'c'를 반환하는 것은?

```
A = ((a, b), c, d)
```

① head(head(A))　　**② head(tail(A))**
③ head(tail(head(A)))　　④ head(tail(tail(A)))

4. 다음은 단순 연결 리스트(singly linked list) L에서 리스트의 원소를 역순으로 바꾸는 함수다. 함수에서 ㉠부분에 들어갈 프로그램 코드(code)로 옳은 것은?

```
class SList:
    class Node:
        def __init__(self, item, link):
            self.item=item
            self.next=link

    def __init__(self):
        self.head=None
        self.size=0

    def reverse(self):
        p=self.head
        q=None
        r=None
        while p is not None :
            r = q
            q = p
            p = p.next
                ㉠
        self.head=q
```

① q = q.next　　**②q.next = r**
③ p.next = q　　④r.next = q

5. 다음 파이썬 함수를 사용하여 function(9)를 수행했을 때, 반환되는 값은?

```
def function(n):
    if n <= 1 :
        return 1
    if n % 2 == 0 :
        return n * function(n-1)
    else :
        return n * function(n-3)
```

① 60　　② 84
③ 540　　④ 67

6. 다음은 Fibonacci 수열을 계산하는 파이썬 함수이다. 함수 fibonacci(4)를 호출하였을 때 화면에 출력되는 숫자의 순서로 옳은 것은?

```
def fibonacci(n) :
    print(n)

    if n == 0 :
        return 0
    elif n == 1 :
        return 1
    else :
        return ( fibonacci( n - 1 ) \
                + fibonacci( n - 2) )
```

① 4, 3, 2, 2, 1, 1, 0, 1, 0
② 4, 3, 2, 1, 0, 2, 1, 1, 0
③ 4, 3, 2, 1, 0, 1, 2, 1, 0
④ 4, 3, 2, 1, 0, 2, 1, 0, 1

7. 다음의 표는 단순 연결 리스트(singly linked list)를 표현한 것으로 각 행은 각 노드의 주소, 데이터 및 다음 노드 주소로 구성되어 있다. <다음 노드 주소 값>은 주소가 102인 노드를 삭제한 후 다음 노드 주소의 값을 설명한 것이다. ㉠ ~ ㉣에 들어갈 값을 바르게 연결한 것은? (단, 특정 노드의 다음 노드가 없을 때 그 노드의 다음 노드 주소 값은 None이다.)

주소	데이터	다음 노드 주소
100	LEE	None
101	PARK	104
102	KIM	101
103	JUNG	102
104	SEO	100

<다음 노드 주소 값>

○ 주소가 100인 노드의 다음 노드 주소는 (㉠)이다.
○ 주소가 101인 노드의 다음 노드 주소는 (㉡)이다.
○ 주소가 103인 노드의 다음 노드 주소는 (㉢)이다.
○ 주소가 104인 노드의 다음 노드 주소는 (㉣)이다.

㉠	㉡	㉢	㉣
① None	**104**	**101**	**100**
② None	103	101	104
③ 101	103	102	104
④ 101	104	102	100

8. 파이썬에서 배열과 연결리스트에 대한 설명으로 옳지 않은 것은?

① 일반적으로 두 개의 정수형 데이터 세트 {1,2,3}과 {4,5,6,7,8}을 합병하여 {1,2,3,4,5,6,7,8}로 만드는 연산을 수행할 경우, 두 데이터 세트의 자료구조를 연결리스트로 만들면, 정수형 배열로 만드는 것보다 시간 복잡도를 낮게 구현할 수 있다.
② 일반적으로 정수형 데이터 세트 {1,2,3,4}를 {4,1,2,3}으로 변경하는 연산을 수행할 경우, 해당 데이터 세트의 자료구조를 연결리스트로 만들면, 정수형 배열로 만드는 것보다 더 빠르게 수행시킬 수 있다.
③ 일반적으로 정수형 데이터 세트 {1,2,3,4}를 {2,3,4,5}로 각 요소에 +1 연산을 수행할 경우, 데이터 세트의 자료구조를 연결리스트로 만들면, 정수형 배열로 만드는 것보다 메모리를 적게 차지한다.
④ 일반적으로 정수형 데이터 세트 {1,2,3,4}를 {0,1,2,3,4}로 새로운 값을 추가하는 연산을 수행할 경우, 데이터 세트의 자료구조를 연결리스트로 만들면, 정수형 배열로 만드는 것보다 더 빠르게 연산을 수행시킬 수 있다.

9. 다음은 단순 연결 리스트(singly linked list)에서 특정 값(value)을 가진 노드의 개수를 반환하는 파이썬 함수이다. ㉠, ㉡에 들어갈 내용으로 옳은 것은?

```
class SList:
    class Node:
        def __init__(self, item, link):
            self.item=item
            self.next=link

    def __init__(self):
        self.head=None
        self.size=0

    def count_node(self, value):
        count=0
        p=self.head
        while _____㉠_____ :
            if p.item == value :
                count+=1
            _____㉡_____
        return count
```

㉠	㉡
① p is None	p.next=p
② p is None	p=p.next
③ p is not None	p.next=p
④ p is not None	**p=p.next**

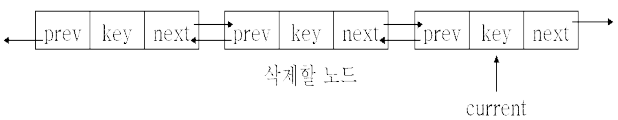
10. 다음은 원형 연결 리스트(circular linked list)의 맨 앞에 insert_node가 지정하는 노드를 삽입하는 파이썬 함수이다. ㉠, ㉡에 들어갈 내용을 바르게 연결한 것은?

```
class CList:
    class _Node :
        def __init__(self, item, link):
            self.item=item
            self.next=link
    def __init__(self):
        self.last=None
        self.size=0

    def no_items(self): return self.size
    def is_empty(self): return self.size==0

    def insertFront(self, item) :
        n=self._Node(item, None)
        if self.is_empty() :
            self.last=n
                    ㉠
        else:
            n.next=self.last.next
                    ㉡
        self.size+=1
```

	㉠	㉡
①	n=n.next	self.last=n
②	n=n.next	self.last.next=n
③	n.next=n	self.last=n
④	**n.next=n**	**self.last.next=n**

11. 다음은 파이썬 클래스와 이를 이용하여 만든 이중 연결 리스트(doubly linked list)이다. current_node가 가리키는 노드의 왼쪽 노드를 삭제할 때 수행할 파이썬 문장을 순서대로 바르게 나열한 것은? (단, 삭제 전에 current_node 노드의 왼쪽에 최소 2개의 노드가 있다고 가정한다.)

```
class Node:
    def __init__(self, item, prev, link):
        self.item = item
        self.prev = prev
        self.next = link
```



① current_node.next=current_node.next.next
current_node.next.prev = current_node
② current_node.next.prev = current_node
current_node.next = current_node.next.next
③ current_node.prev = current_node.prev.prev
current_node.prev.next = current_node
④ current_node.prev.next = current_node
current_node.prev = current_node.prev.prev

12. 입력받은 수식의 괄호쌍이 맞는지 스택 S를 이용하여 판단하는 알고리즘 DDD를 작성하였다. 수식에 존재하는 괄호는 소괄호 '()'이며, 다른 괄호는 수식에 존재하지 않는다. 수식의 길이는 n이며, 배열 X에 저장되어 입력된다. 알고리즘 DDD는 X를 입력받아 X에 저장된 수식의 괄호쌍이 맞으면 True, 그렇지 않으면 False를 반환하는 함수이다. ㉠ ~ ㉢에 들어갈 코드를 바르게 연결한 것은? (단, $S.push(c)$는 스택 S에 문자 c를 삽입하는 연산, $S.pop()$은 S에서 삭제하는 연산, $S.empty()$는 S가 비어 있으면 True, 아니면 False를 반환하는 함수이다)

```
Algorithm DDD(X, n)
Let S be an empty stack
for i←0 to n−1 do
  if X[i] = '(' then
    S.push(X[i])
  else if X[i] = ')' then
    if S.empty() then
          ㉠
    end if
    S.pop()
  end if
end for
if S.empty() then
      ㉡
else
      ㉢
end if
```

	㉠	㉡	㉢
①	return True	return True	return False
②	**return False**	**return True**	**return False**
③	return True	return False	return True
④	return False	return False	return True

13. 다음과 같이 중위 표기법(infix notation)으로 되어 있는 수식을 후위 표기법(postfix notation)으로 변환하기 위해 스택을 이용한다. 변환 과정에서 스택에 토큰이 가장 많이 쌓여 있을 때의 스택 내 연산자 토큰 수는?

a*(b-c*d)/e

① 3개 ② 4개
③ 5개 ④ 6개

14. <조건>을 만족하고 front와 rear의 값이 모두 0인 원형 큐(circular queue)에서 <동작>에 나열된 연산을 순서대로 모두 수행하였다. 연산 완료 후 원형 큐에 존재하는 모든 유효 데이터와 front 및 rear의 값을 바르게 연결한 것은? (단, enqueue(*x*)에 의해 삽입된 후 dequeue()에 의해 삭제되지 않은 데이터만 유효 데이터로 인정한다)

<조 건>

○ 원형 큐는 원소 개수가 6개인 1차원 배열로 구현되었다.
○ front와 rear의 값이 같을 때 원형 큐는 공백 상태이다.
○ front와 ((rear + 1) mod 6)의 값이 같을 때 원형 큐는 포화 상태이다.
○ enqueue(*x*)에 의해 rear의 값을 ((rear + 1) mod 6)로 변경한 후 rear가 가리키는 위치에 데이터 *x*를 삽입한다. 단, 포화 상태일 때에는 데이터 삽입이 이루어지지 않아 원형 큐의 상태가 변하지 않는다.
○ dequeue()에 의해 front의 값을 ((front + 1) mod 6)로 변경한 후 front가 가리키는 위치에서 데이터를 빼내어 삭제한다. 단, 공백 상태일 때에는 데이터 삭제가 이루어지지 않아 원형 큐의 상태가 변하지 않는다.

<동 작>

enqueue(2) → enqueue(5) → enqueue(7) → enqueue(10) → dequeue() → enqueue(11) → enqueue(15) → enqueue(17) → dequeue() → dequeue() → enqueue(20)

모든 유효 데이터	front	rear
① 10, 11, 15, 17, 20	3	1
② 10, 11, 15, 20	3	1
③ 10, 11, 15, 17, 20	4	2
④ 10, 11, 15, 20	4	2

15. 스택을 이용하여 다음 후위 표현식(postfix expression)의 연산을 수행하였다. 최종 결과 값과 연산 과정에서 스택 내 피연산자의 최대 개수를 바르게 연결한 것은? (단, *은 곱셈 연산자이고 /은 나눗셈 연산자이다)

9 1 − 2 / 1 2 1 + * − 1 −

	최종 결과 값	스택 내 피연산자의 최대 개수
①	0	3
②	0	4
③	1	3
④	1	4

16. 다음은 구현하고자 하는 원형 큐(circular queue)에 대한 <조건>과 <설명>이다. <설명>의 괄호 안에 들어갈 수로 옳게 짝지어진 것은?

<조 건>

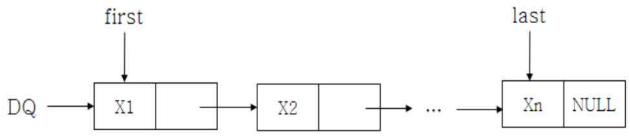
○ 크기가 6이고 인덱스가 0부터 5까지인 배열로 구현된다.
○ 공백 상태에서만 front와 rear의 값이 서로 같으며, 그 외에는 값이 서로 다르다.
○ (front + 1) mod6은 다음에 삭제할 원소 위치를 가리킨다.
○ (rear + 1) mod6은 다음에 삽입할 원소 위치를 가리킨다.

<설 명>

○ 최대 (㉠)개의 원소를 저장할 수 있다.
○ front=5이고 rear=3인 경우 저장된 원소 수는 (㉡)개이다.
○ front= (㉢)이고 rear=2인 경우 포화 상태를 의미한다.

㉠	㉡	㉢
① 6	5	1
② 6	4	1
③ 5	5	3
④ 5	4	3

17. 데크(deque: double-ended queue)는 삽입과 삭제가 양끝에서 임의로 수행되는 자료구조이다. 다음 그림과 같이 단순 연결리스트(singly linked list)로 데크를 구현한다고 할 때 O(1) 시간 내에 수행할 수 없는 연산은? (단, first와 last는 각각 데크의 첫 번째 원소와 마지막 원소를 가리키며, 연산이 수행된 후에도 데크의 원형이 유지되어야 한다)



① insertFirst 연산: 데크의 첫 번째 원소로 삽입
② insertLast 연산: 데크의 마지막 원소로 삽입
③ deleteFirst 연산: 데크의 첫 번째 원소를 삭제
④ deleteLast 연산: 데크의 마지막 원소를 삭제

18. 큐(queue)를 적용할 수 있는 상황이 아닌 것은?

① 도착한 순서대로 프린트 처리

② 비동기적 데이터 전송 (파일 입·출력, 파이프, 소켓)

③ 프로세스 스케줄링

④ 텍스트 에디터에서의 작업 취소(undo) 과정

19. 다음의 해시 함수을 사용하는 이중 해싱(double hashing)에서 해시 테이블의 크기는 7이며 0부터 6까지의 인덱스를 가진다. $h(x)$는 첫 번째 조사 위치를 결정하는 기본적인 해시 함수이고, $f(x)$는 충돌 발생 시 조사 위치 간격을 결정하는 추가 해시 함수로서, i번째 충돌 발생 시 다음 조사 위치를 결정하는 해시 함수는 $h_i(x)$가 된다. 공백 해시 테이블에 일련의 키(key) 값 9, 10, 2, 3, 16, 13, 11을 가지는 레코드들을 순서대로 삽입한 후, 해시 테이블의 위치별 키 값을 보여주는 것은? (단, 해시 테이블 버킷(bucket)당 슬롯(slot) 수는 1개이다)

$$h(x) = x \bmod 7$$
$$f(x) = 5 - (x \bmod 5)$$
$$h_i(x) = (h(x) + i \times f(x)) \bmod 7$$

	[0]	[1]	[2]	[3]	[4]	[5]	[6]
①	13	16	9	10	11	2	3
②	16	13	9	10	11	2	3
③	**3**	**13**	**9**	**10**	**11**	**2**	**16**
④	13	11	9	10	2	3	16

20. 해시 테이블 HT의 크기는 11이고 0부터 10까지의 인덱스를 가진다. 해시 함수 h(x)는 제산 함수로서 h(x) = xmod 11로 표현된다. 충돌 해결책으로는 이차 조사법(quadratic probing)을 사용한다. 해싱을 통해 해시 테이블에 저장할 레코드들의 키 값 순서는 다음과 같다. 마지막 레코드가 저장되는 해시 테이블 위치는? (단, 첫 번째 레코드가 저장되기 전에 해시 테이블은 비어있고, 해시 테이블 버킷(bucket)당 슬롯(slot) 수는 1개이다)

51, 27, 70, 55, 13, 2, 37, 23, 33

① HT[1]　② HT[4]

③ HT[6]　**④ HT[9]**

21. key 값이 각각 23, 42, 12, 56, 62, 79, 3인 입력 레코드가 순차적으로 삽입되고 해시함수가 h(key) = key mod 11로 정의된 상태에서 충돌(collision)이 발생하였을 경우 이를 해결하기 위한 충돌 해결법(collision resolution)에 관한 다음의 물음에 답하시오. (단, 아래의 같이 해시 테이블에 관한 배열의 크기는 11이고, 해시함수 h(key)에서 mod는 나머지 연산자이다.)

(1) 선형탐사(linear probing)를 수행한 결과를 아래 배열에 기록하고, 선형탐사의 문제점을 설명하시오. (6점)

(배열이 다 맞으면 2점)

index	0	1	2	3	4	5	6	7	8	9	10
key		**23**	**12**	**56**	**79**	**3**		**62**		**42**	

[문제점]

(키워드 하나당 2점 총 4점)

- 선형탐사는 순차적으로 탐사하여 빈 bucket를 찾아 충돌된 key를 저장하므로 해시테이블의 key들이 빈틈없이 뭉쳐지는 현상이 발생[1차 군집화(primary clustering)]함. 이러한 군집화는 삽입, 삭제, 탐색 연산 시 군집된 key들을 순차적으로 방문해야 하는 문제점을 야기. 탐색의 시간을 좌우하는 건 key 값이 속한 군집의 크기에 좌우됨.

(2) 이차탐사(quadratic probing)를 수행한 결과를 아래 배열에 기록하고, 이차탐사의 문제점을 설명하시오. (9점)

(배열이 다 맞으면 5점)

index	0	1	2	3	4	5	6	7	8	9	10
key		**23**	**12**	**79**	**3**	**56**		**62**		**42**	

[문제점]

(키워드 하나당 2점 총 4점)

- 같은 해시값을 갖는 서로 다른 key들인 동의어(synonym)들이 똑같은 점프 시퀀스(jump sequence)를 따라 빈 bucket를 찾아 저장하므로 결국 또 다른 형태의 군집화인 2차 군집화(secondary clustering)를 야기

- 점프 크기가 제곱 만큼씩 커지므로 배열에 빈 bucket가 있는데도 빈 bucket를 건너뛰어 저장에 실패하는 경우도 피할 수 없음

(3) 보조해시함수가 d(key) = 7 - (key mod 7)로 정의된 이중해싱(double hashing)을 수행한 결과를 아래 배열에 기록하고, 이중해싱의 문제점을 설명하시오. (9점)

(배열이 다 맞으면 7점)

index	0	1	2	3	4	5	6	7	8	9	10
key	3	23	79	12				62	56	42	

[문제점]

(키워드 들어가면 2점)

- 선형탐사나 이차탐사에 비해 다음 점프(next probe)를 계산하는데 더 많은 비용이 듦 (그러나 큰 군집보다 바람직함)

(4) 입력 레코드의 삽입이 계속되면 적재율(load factor)의 한계에 부딪혀 더 큰 해시테이블을 할당받아 (M → M′), 기존의 해시테이블의 값을 모두 옮겨야 한다. N >= M/2이 될 때, M′ = 2M로 테이블 크기를 2배 늘리는 방법을 사용한다고 할 때 N 번의 삽입이 연속적으로 일어나면서 테이블 크기 조정이 여러 번 발생하는 최악의 상황을 생각해보자. 이때의 한 번의 삽입 연산의 평균 수행시간을 amortized analysis로 설명하시오. (6점)

1. N번의 삽입과 이에 따른 테이블 크기 M이 재조정 (1점)
   $M = 2 \rightarrow 2^2 \rightarrow 2^3 \rightarrow \cdots \rightarrow 2^k$
   $N = 1 \rightarrow 2^1 \rightarrow 2^2 \rightarrow \cdots \rightarrow 2^{k-1}$

2. k번의 크기 조정이 일어날 때마다 발생하는 비용 c (1점)
   $c = 1 \rightarrow 2^1 \rightarrow 2^2 \rightarrow \cdots \rightarrow 2^{k-1}$

3. 총비용은 $1 + 2^1 + \cdots + 2^{k-1} = 2^k - 1$ 이 된다 (2점)

4. $N = 2^{k-1}$ 이므로 평균 비용은 $(2^k - 1)/2^{k-1} < 2$이 된다. 즉, amortized analysis로 따지면 한 번의 삽입 연산의 평균 수행시간은 O(1)이다. (2점)