IDEAOA01: Algorithm is a step-by-step procedure for solving a problem in a finite amount of time

IDEAOA02

● The execution time and the memory needed for an algorithm must be precisely defined.

IDEAOA03.

○ A data structure is a piece of information (a physical instantiation of a data type)

IDEAOA04.

○ The complexity of an algorithm is a measure of the amount of time and space required by the algorithm for an input of a given size n.

IDEAOA05

○ Theoretical approach

IDEAOA06

○ The amount of time that the algorithm needs to run for an input of a given size n.

IDEAOA08

● The best-case is used frequently to analyze the time complexity of algorithms.

IDEAOA09

○ The average-case is easy to determine.

IDEAOA10

○ At the worst-case the algorithm takes more time to finish than it does at the average-case and best-case.

IDEAOA11

● The leading term.

IDEAOA12

● Big-Oh notation

IDEAOA13

○ Big-Omega notation

IDEAOA14

● We compare the grow rate of the leading terms of TA(N) and TB(N).

IDEAOA15: They are mathematic notation for comparing growth rates between functions

IDELI01

○ List can be implemented using an array or a collection of linked nodes.

IDELI02

○ Integer.

IDELI03

● Boolean.

IDELI04

● Array-based is faster than linked-list in case of accessing list’s items.

IDELI05

○ Elements of linked-list can be located dynamically and discontinuously.

IDELI06

○ Remove an item at the pos position form the list.

IDELI07

○ null

IDELI08

○ null

IDELI09

● null.

IDELI10

○ X.getNext().getNext().

IDELI11

● Y.getNext().

IDELI12

○ c=c+1

IDELI13

○ head

IDELI14

● X(data, prev, next)

IDELI15

○ X(data, next)

IDELI16

● Abstract Data Type.

IDESOA01: The sort key must be numeric.

IDESOA02

○ The time complexity of some comparison sorting algorithms can be faster than O(NlogN).

IDESOA03

● The sorted order is determined based on the comparisons between sort keys.

IDESOA04

● The relative order of elements with equal keys are maintained.

IDESOA05

○ Based on Divide and Conquer approach.

IDESOA06

● Selection sort.

IDESOA07

○ We must shift several elements to make place for the inserted one.

IDESOA08

○ Insertion sort.

IDESOA09

○ In a min-heap the parent node value is always greater than or equal to its children's values.

IDESOA10

○ .The input array is divided into two parts at the middle of the array

IDESOA011

● A merge algorithm is needed to combine two partitioned arrays

IDESOA12: A stable sorting algorithm is used to sort the digits.

IDESOA13

It is an internal sorting algorithm.

IDESOA14

● O(P(N+B)).

IDESOA15

● Sorting.

IDESOA16

○ Merge sort.

IDESQ01

● It is a First In First Out (FIFO) list.

IDESQ02

○ enqueue() and dequeue() operations must be performed at one end of the queue.

IDESQ03

○ add an item to the stack

IDESQ04

○ top is the last item of the array.

IDESQ06

○ Remove an item from the queue at the front position.

IDESQ07

● Add a new item to the queue at the rear position.

IDESQ08

● when front=rear the queue is full.

IDESQ09

● rear=(rear+1)%maxSize

IDESQ10

● front=(front+1)%maxSize

IDESQ11

○ Queue is empty when front=rear.

IDESQ12

○ The contents of a queue can wrap around , while those of a stack can not

IDESQ13

○ 10

IDESQ14

● 40

IDESQ15

● Rear

IDESQ16

● X.

IDESQAS01

○ Linear time.

IDESQAS03

○ Constant time.

IDESQAS04

● The array must be sorted.

IDESQAS05

● Two entries with different keys have the same exact hash value.

IDESQAS06

○ 512.

IDESQAS08

○ Evaluating a posfix expression

IDESQAS10

○ 6.

IDESQAS11

● i.

IDESQAS12

○ hi(k)=(h(k)+i) mod N.

IDESQAS13

● 4199 and 9679 hash to the same value.

IDESQAS14

● Binary search is faster than linear search, but it requires a sorted array.

IDESQAS15

● middle=(left + right)/2

IDHLI01

○ Insertion sort.

IDHLI02

○ head=prev

IDHLI03

● O(1) and O(N)

IDHLI04

● beforeTail.setNext(null)

IDHLI05

● beforeTail.setNext(null); tail.setNext(head); head=tail;

IDHSOA01

○ The array is already sorted in the ascending order.

IDHSOA02

● The array is already sorted in the descending order.

IDHSOA03

○ Insertion sort

IDHSOA04

● Insertion sort

IDHSOA05

□ T(N)=T(N/3)+T(2N/3)+O(N)

IDHSOA06

● Merge sort

IDHSOA07

○ Radix sort

IDHSQAS01

● Stack.

IDHSQAS03

○ Quadratic probing

IDHSQAS04

○ Less than 1.

IDHSQAS05

● Stack.

IDMAOA01

○ O(N^2)

IDMAOA02

● O(N)

IDMAOA03

● O(N)

IDMAOA04: O(N)

IDMAOA05

● O(N)

IDMAOA06

○ O(N^3)

IDMAOA07

○ O(N)

IDMAOA08

● O(N^3)

IDMAOA09

○ O(N^2)

IDMAOA10

● O(N^3)

IDMAOA11

○ O(N^5)

IDMAOA12

○ Two algorithm are equivalent in term of time efficiency.

IDMAOA13

○ Algorithm 3, Algorithm 1, Algorithm 2

IDMAOA14

● Algorithm 2, Algorithm 1, Algorithm 3

IDMAOA15

○ O(N^2)

IDMLI02

○ Delete all nodes from the list.

IDMLI03

● Remove an item from the list.

IDMLI04

● Remove the node at the pos position from the list

IDMLI05

● Insert an item to the list

IDMLI06

● ‘A’-->‘B’-->‘C’-->‘D’-->‘E’-->‘F’

IDMLI07

○ ‘F’-->‘E’-->‘D’-->‘C’-->‘B’-->‘A’

IDMLI08

○ ‘A’-->‘C’-->‘E’

IDMLI09

○ ‘E’-->‘C’-->‘A’

IDMLI10

- Consider method F in Java and a singly linked list L below. Suppose that H is the head node of

● ‘B’-->‘D’-->‘F’

IDMLI11

● ‘F’-->‘D’-->‘B’

IDMLI12

● N

IDMSOA01

○ A={23,32,8,45,56,78}

IDMSOA02

● A={8,23,45,78,32,56}

IDMSOA03

○ A={8,23,32,45,56,78} (đáp án đúng là {8, 23,32,45,78,56 } cái này gần nhất nên chắc là ok :3)

IDMSOA04: Merge sort

IDMSOA05

○ Selection sort

IDMSOA06

● Bubble sort

IDMSOA08

○ A={2,5,9,8,10,13,12,22,50}

IDMSOA09

● A={78,56,45,32,23,8,15}

IDMSOA10

○ C={0,1,3,4,4,5}

IDMSOA11

○ A={900,802,145,170,275}

IDMSOA12

○ C={3,9,10,27,38,43,82}

IDMSOA13

○ A={19,17,16,7,15,10}

IDMSOA14

○ A={10,22,2,9,30,42,52,33}

IDMSQ01

● S={“A”,“B”,“C”,“D”}

IDMSQ02

○ Q={“D”, “E” ,“F” ,“D”}

IDMSQ03

● pop()-->pop()-->pop()-->push(“2”)-->push(“3”)-->push(“1”)

IDMSQ04

○ enqueue(“4”)-->enqueue(“5”)-->dequeue()-->dequeue()

IDMSQ05

○ erutcurtsatad

IDMSQ07

● Queues use two ends of the structure; stacks use only one.

IDMSQ08

○ No operation that has time complexity O(N).

IDMSQ09

○ No operation that has time complexity O(N).

IDMSQ10

○ m=m-1

IDMSQ11

● m=m-1

IDMSQ12

○ 5

IDMSQ13

○ Simple array-based queue.

IDMSQAS01

○ 8 2 + 5 7 +* 10 -9*3+

IDMSQAS02

● + 5 * + 7 * 9 3+ 2 8

IDMSQAS03

○ 129

IDMSQAS04

○ 150

IDMSQAS05

○ Print binary representation of n.

○ 3

IDMSQAS07

○ 6

IDMSQAS08

● B

IDMSQAS09

○ A

IDMSQAS10

● 1, 8, 10, -, -, -, 3

IDMSQAS11

● Order of the elements of the list.

IDHAOA03 – What is O(f(N)) if

● O(N^3)

IDHAOA04

○ O(N^6) -> đáp án sai

IDHAOA05: O(N)

IDESQAS09 Message buffering.

IDHAOA07 O(N^4)

IDHSQAS02 The task cannot be accomplished.

IDHAOA10 :O(2^N)

IDHAOA08: O(N^4)

1. **IDMTRE01**: Descending order.
2. IDMTRE06: A,B,D,C,E,G,J,F,H,I

3. IDMTRE07: B,D,A,G,J,E,C,H,F,I
4. IDMTRE08: D,B,J,G,E,H,I,F,C,A
5. IDMTRE10: 3
6. IDMTRE12: Min heap.
7. IDMTRE13: 4
8. IDMTRE17: Value of node C is smaller than value of node A and node B.
9. IDMTRE18: Value of node C is bigger than value of node B, but smaller than value of node A.
10. IDMTRE20: Node C has the biggest value
11. IDMTRE21: 21

12. **IDMGRA01**: N-1
13. IDMGRA02: Parallel edges
14. IDMGRA03: Performing a BFS starting from S
15. IDMGRA04: P, Q, R, U, S, T

16. **IDEGRA01**: 2E.
17. IDEGRA02: Queue
18. IDEGRA03: The weight of the shortest path from vertex Vi to vertex Vj using intermediate verties in the set {V1..Vk}.
19. IDEGRA04: Parallel edges.
20. IDEGRA05: A symmetric matrix over its diagonal.
21. IDEGRA06: A matrix contains only 0 and 1.
22. IDEGRA07: Unweighted, undirected, complete graph
23. IDEGRA08: Weigh of an edge must be possitive.
24. IDEGRA09: Queue
25. IDEGRA10: Adding a vertex in adjacency matrix representation is easier than adjacency list representation.

26. **IDETRE03**: Complete binary tree.
27. IDETRE04: 2^h.
28. IDETRE07: This is a binary search tree.

29. IDETRE08: This is an expression tree.
30. IDETRE12: Node C.
31. IDETRE13: Node G.
32. IDETRE14: The parent node of node K.
33. IDETRE15: p[node]
34. IDETRE16: l[node]
35. IDETRE17: The left child and right child of node i are 2i+1 and 2i+2
36. IDETRE19 : preOrderTraversal(getLeftChild(node))
37. IDETRE21: postOrderTraversal(getRightChild(node))
38. IDETRE22: t.getRightSubTree()
39. IDETRE23: t.getLeftSubTree()

40. **IDHTRE01**: Post-order
41. IDHTRE02: One node.
42. IDHTRE06: (1 (2 3 4) (5 6 7))
43. IDHTRE07: E
44. IDHTRE10: DECBUTZYXA

45. The method below represent a number k in base b using a stack. Please complete the code of this method? (đúng ½ code -_-)

```
public void BaseConversion(int k, int b)
{
      ArrayStack s = new ArrayStack();
      while (k/b != 0)
      {
          s.push(k%b);
          [k=k/b;]   ; k=k/b;
      }
      s.push(k);
      while (!s.isEmpty())
          System.out.print([s.pop()]  ); s.pop()
}
```

46. Method search() is used to search for an item in a singly linked list. Please complete the code for this method?

```
    public int search(int data)
    {
        int count=1;
        SLNode current=this.head;
```

```
            while ((current !=null) && (current.getData()
!=  data  )) data
            {
                count++;
                current= current.getNext() ; current.getNext()
            }
            if (current == null)
                return -1;
```

```
        else

            return  count  ;   count

    }
```

47. The following method reverses the item's order of a stack using a queue. Please complete the code of the method?

```
    public static int reverse(SLLStack s)
    {
            ArrayQueue q = new ArrayQueue();
            while (!s.isEmpty())
            {
                StackNode node
=   s.pop()  ;     s.pop()
                q.enqueue(node.getData());
            }
            while ( !q.isEmpty() ) !q.isEmpty()
            {
                StackNode newnode = new StackNode(q.dequeue());
                s.push( new noc ); newnode
            }
    }
```

48. The following method reverses the item's order of a stack using a queue. Please complete the code of the method?

```
    public static int reverse(SLLStack s)
    {
            ArrayQueue q = new ArrayQueue();
            while (!s.isEmpty())
            {
            StackNode node =  s.pop()  ; s.pop()
```

```
                q.enqueue(node.getData());
            }
            while ( !q.isEmpty() ) !q.isEmpty()
            {
                StackNode newnode = new StackNode(q.dequeue());
                s.push( new noc ); newnode
            }
    }
```

49. This method implement an O(N) algorithm to rearrange array x so that the left part is the elements that is smaller than p, the right part is the elements that is bigger than p. Please complete the code for this method?

```
public static void rearrange(int [] x, int p)
{
int left=0;
int right=x.length-1;
while ( left<right ) left<right
{
while ((x[left]<p)&&(left<x.length))
 left++ ; left++
while ((x[right]>p)&&(right>=0))
 right-- ; right--
if (left<right)
{
int tmp=x[left];
x[left]=x[right];
x[right]=tmp;
}
}
}
```

50. Method search() is used to search for an item in a singly linked list. Please complete the code for this method?

```
    public int search(int data)
    {
        int count=1;
         SLNode current=this.head;
        while ((current !=null) && (current.getData()
!=  data  )) data
        {
            count++;
            current= current.getNext()  ; current.getNext()
        }
        if (current == null)
            return -1;
        else
            return  count  ;  count
    }
```

51. Method search() is used to search for an item in a singly linked list. Please complete the code for this method?

```
public int search(int data)
{
        int l=getLength();
        for (int i=1; i<l; i++)
        {
            SLNode aNode= aNode  ; aNode.get(i)
            if (aNode.getData()==data)
                return i;
        }
        return  0  ; 0
}
```

52. The following method implement the recursive version of the binary search algorithm. Please complete the code of the method?

```
    public static int BinarySearch(int []a, int key, int
left, int right)
    {
        if (left > right)
            return KE  ; return  KEY_NOT_FOUND
        else
        {
            int mid = (left + right)/2;
            if ( a[mid]<key  ) a[mid]<key
               return BinarySearch(a,key,mid+1,right);
            else
            {
               if (a[mid]>key)
               return BinarySearch(a,key,left, mid-1  ); mid-1
               else
```

```
                    return mid;
               }
           }
       }
```

53. Please complete the code of the linear search method below?

```
  public int LinearSearch(int[] a, int key)
  {
        int index=0;
        boolean found=false;
        int pos=-1;
        while ((index<n)&&(!found))
        {
            if ( a[index]!=key ) a[index]!=key
                        {
                             found=true;
                             pos=index;
                        }
                        index++;
        }
                return pos ;pos
  }
```

54. Method swap() is used to swap two nodes in a Singly Linked List. Please complete the code for this method?

```
    public void swap(int pos1, int pos2)

    {

        SLNode node1 = get(pos1);

        SLNode node2 = get(pos2) ; get(pos2)

        SLNode tmp=new SLNode(node1.getData());

        node1.setData( node2.getData() );node2.getData()

        node2.setData( tmp.getData() );  tmp.getData()
```