Elementary Data Structures and Algorithms

Arrays

Always choose the best or most general answer, unless otherwise instructed.

Concept: *simple arrays*

Assume zero-based indexing for all arrays.

In the pseudocode, the lower limit of a `for` loop is inclusive, while the upper limit is exclusive. The step, if not specified, is one.

For all types of fillable arrays, the size is the number of elements added to the array; the capacity is the maximum number of elements that can be added to the array.

1. Consider a small array *a* and large array *b*. Accessing the element in the first slot of *a* takes more/less/the same amount of time as accessing the element in the first slot of *b*.

    A. it depends on how the arrays were allocated

    B. less time

    C. more time

    D. the same amount of time

2. Consider a small array *a* and large array *b*. Accessing the element in the last slot of a *b* takes more than/less than/the same amount of time as accessing an element in the middle slot of *a*. Both indices are supplied.

    A. more time

    B. the same amount of time

    C. less time

    D. it depends on how the arrays were allocated

3. Accessing the middle element of an array takes more/less/the same amount of time than accessing the last element.

    A. it depends on how the array were allocated

    B. the same amount of time

    C. more time

    D. less time

4. What is a major characteristic of a simple array?

    A. finding an element can be done in constant time

    B. getting the value at an index can be done in constant time

C. inserting an element between indices *i* and *i+1* can be done in constant time

5. What is a *not* a major characteristic of a simple array?

   A. getting the value at an index can be done in constant time

   B. swapping two elements can be done in constant time

   C. finding an element can be done in constant time

   D. setting the value at an index can be done in constant time

6. Does the following code set the variable *v* to the minimum value in an unsorted array with at least two elements?

```
v = 0;
for (i from 0 until array.length)
    if (array[i] < v)
        v = array[i];
```

   A. only if the true minimum value is zero

   B. only if all elements have the same value

   C. yes, if all the elements are negative

   D. never

   E. yes, if all the elements are positive

   F. always

7. Does the following code set the variable *v* to the minimum value in an unsorted array with at least two elements?

```
v = array[0];
for (i from 0 until array.length)
    if (array[i] < v)
        v = array[i];
```

   A. only if all elements have the same value

   B. yes, if all the elements are positive

   C. only if the true minimum value is at index 0

   D. always

   E. never

   F. yes, if all the elements are negative

8. Does the following code set the variable *v* to the minimum value in an unsorted array with at least two elements?

```
v = array[0];
for (i from 0 until array.length)
    if (array[i] > v)
        v = array[i];
```

   A. only if the true minimum value is at index 0

   B. only if all elements have the same value

   C. never

   D. yes, if all the elements are negative

   E. yes, if all the elements are positive

   F. always

9. Does the following code set the variable *v* to the minimum value in an unsorted, non-empty array?

```
v = array[0];
for (i from 0 until array.length)
    if (array[i] > v)
        v = array[i];
```

   A. yes, if all the elements are positive

   B. only if all elements have the same value

   C. only if the true minimum value is at index 0

   D. always

   E. yes, if all the elements are negative

   F. never

10. Does this *find* function return the expected result? Assume the array has at least two elements.

```
function find(array,item)
    {
    var i; var found = False;
    for (i from 0 until array.length)
        if (array[i] == item)
            found = True;
    return found;
    }
```

   A. never

   B. only if the item is not in the array

   C. only if the item is in the array

   D. always

11. Does this *find* function return the expected result? Assume the array has at least two elements.

```
function find(array,item)
    {
    var i;
    for (i from 0 until array.length)
        if (array[i] == item)
            return False;
    return True;
    }
```

   A. always

   B. never

   C. only if the item is in the array

   D. only if the item is not in the array

12. Is this *find* function correct? Assume the array has at least two elements.

```
function find(array,item)
    {
    var i; var found = True;
    for (i from 0 until array.length)
        if (array[i] != item)
            found = False;
    return found;
    }
```

A. always

B. only if the item is not in the array

C. never

D. only if the item is in the array

13. Does this *find* function return the expected result? Assume the array has at least two elements.

```
function find(array,item)
    {
    var i;
    for (i from 0 until array.length)
        if (array[i] == item)
            return True;
    return False;
    }
```

A. never

B. only if the item is in the array

C. only if the item is not in the array

D. always

Concept: *simple fillable arrays*

Assume the back index in a simple fillable array points to the first available slot.

14. What is *not* a property of a simple fillable array?

A. elements can be added in constant time

B. there exists an element that can be removed in constant time

C. the underlying simple array can increase in size

D. elements are presumed to be contiguous

15. What is a property of a simple fillable array?

A. elements are presumed to be contiguous

B. an element can be added anywhere in constant time

C. any element can be removed in constant time

D. more that one element can be next to an empty slot

16. Adding an element at back of a simple fillable array can be done in:

A. linear time

B. constant time

C. quadratic time

D. logarithmic time

17. Removing an element at front of a simple fillable array can be done in:

A. logarithmic time

B. constant time

C. quadratic time

D. linear time

18. Suppose a simple fillable array has size s and capacity c. The next value to be added to the array will be placed at index:

A. *c*

B. *s* + 1

C. *c* + 1

D. *s*

E. *c* - 1

F. *s* - 1

19. Suppose for a simple fillable array, the size is one less than the capacity. How many values can still be added?

    A. this situation cannot exist

    B. zero, the array is full

    C. one

    D. two

20. Suppose for a simple fillable array, the capacity is one less than the size. How many values can still be added?

    A. zero, the array is full

    B. one

    C. two

    D. this situation cannot exist

21. Suppose a simple fillable array is empty. The size of the array is:

    A. the length of the underlying simple array

    B. the capacity of the array

    C. zero

    D. one

22. Suppose a simple fillable array is full. The capacity of the array is:

    A. zero

    B. one

    C. the length of the underlying simple array

    D. its size minus one

23. Which code fragment correctly inserts a new element into index *j* of a simple fillable array with size *s*? Assume there is room for the new element.

```
for (i from j until s-2)
    array[i] = array[i+1];
array[i] = newElement;
---
for (i from s-2 until j)
    array[i+1] = array[i];
array[i] = newElement;
```

    A. the second fragment

    B. neither are correct

    C. both are correct

    D. the first fragment

24. Which code fragment correctly inserts a new element into index *j* of an array with size *s*?

```
for (i from j until s-2)
    array[i+1] = array[i];
array[i] = newElement;
---
for (i from s-2 until j)
    array[i] = array[i+1];
array[i] = newElement;
```

A. neither are correct

B. the second fragment

C. both are correct

D. the first fragment

Concept: *circular arrays*

For circular arrays, assume f is the start index, e is the end index, s is the size, and c is the capacity of the array. Both f and e point to the first available slots.

25. What is a property of a theoretical (not practical) circular array?

    A. any element can be removed in constant time

    B. an element can be added anywhere in constant time

    C. there are two places an element can be added

    D. elements do not have to be contiguous

26. What is *not* a property of a theoretical (not practical) circular array?

    A. elements are presumed to be contiguous

    B. appending an element can be done in constant time

    C. inserting an element in the middle can be done in constant time

    D. prepending an element can be done in constant time

27. The next value to be added to the front of a circular array will be placed at index:

    A. c - 1

    B. $s + f$

    C. $c - f$

    D. $s - f$

    E. f

    F. f - 1

28. Suppose for a circular array, the size is equal to the capacity. Can a value be added?

    A. No, the array is completely full

    B. Yes, there is room for one more value

29. Suppose a circular array is empty. The size of the array is:

    A. one

    B. the length of the array

    C. the capacity of the array

    D. zero

30. In a circular array, which is *not* a proper way to correct the start index *f* after an element is added to the front of the array?

    A. `f = (f - 1 + c) % c;`

    B. `f -= 1; f == 0? c - 1 : f;`

    C. `f = f == 0? c - 1 : f - 1;`

    D. `f -= 1; f = f < 0? c - 1 : f;`

    E. `if (f == 0) f = c - 1; else f = f - 1;`

31. **T** or **F**: In a circular array, the start index (after correction) can never equal the size of the array.

32. **T** or **F**: In a circular array, the start index (after correction) can never equal the capacity of the array.

33. Is a separate end index *e* needed in a circular array?

    A. no, it can be computed from *s* and *c*.

    B. no, it can be computed from *s* and *f*.

    C. yes

    D. no, it can be computed from *s*, *c*, and *f*.

    E. no, it can be computed from *c* and *f*.

Concept: *dynamic arrays*

34. What is *not* a major characteristic of a dynamic array?

    A. finding an element takes at most linear time

    B. the array can grow to accommodate more elements

    C. elements are presumed to be contiguous

    D. the only allowed way to grow is doubling the size

    E. inserting an element in the middle takes linear time

35. Suppose a dynamic array has size *s* and capacity *c*, with *s* equal to *c*. Is the array required to grow on the next addition?

    A. yes

    B. no

    C. yes, but only if the dynamic array is not circular

36. Suppose array capacity grows by 50% every time a dynamic array fills. If the only events are insertions, the growing events:

    A. occur more and more frequently

    B. cannot be characterized in terms of frequency

    C. occur periodically

    D. occur less and less frequently

37. Suppose array capacity doubles every time a dynamic array fills, If the only events are insertions, the average cost of an insertion, in the limit, is:

    A. the log of the capacity

    B. the log of the size

    C. constant

    D. linear

38. Suppose array capacity grows by 10 every time a dynamic array fills, If the only events are insertions, the average cost of an insertion, in the limit, is:

    A. constant

    B. quadratic

    C. the log of the size

    D. linear

    E. the log of the capacity

39. Suppose array capacity grows by 10 every time a dynamic array fills, If the only events are insertions, the growing events:

    A. cannot be characterized in terms of frequency

    B. occur periodically

    C. occur less and less frequently

    D. occur more and more frequently

40. If array capacity grows by 10 every time a dynamic array fills, the average cost of an insertion in the limit is:

    A. constant

    B. the log of the size

    C. the log of the capacity

    D. linear