Elementary Data Structures and Algorithms

Linked Lists

Concept: *singly-linked lists (insertions)*

1. Appending to a singly-linked list without a tail pointer takes:

    A. linear time

    B. log time

    C. n log n time

    D. constant time

2. Appending to a singly-linked list with a tail pointer takes:

    A. linear time

    B. n log n time

    C. constant time

    D. log time

3. Suppose you have a pointer to a node near the end of a long singly-linked list. You can then insert a new node just prior in:

    A. linear time

    B. constant time

    C. n log n time

    D. log time

4. Suppose you have a pointer to a node near the end of a long singly-linked list. You can then insert a new node just after in:

    A. n log n time

    B. log time

    C. constant time

    D. linear time

5. Suppose you have a pointer to a node near the end of a long singly-linked list. You can then insert a new node just after with as few pointer assignments as:

    A. 1

    B. 2

    C. 4

    D. 5

    E. 3

Concept: *singly-linked lists (deletions)*

6. Removing the first item from a singly-linked list without a tail pointer takes:

A. constant time

B. log time

C. *n* log *n* time

D. linear time

7. Removing the last item from a singly-linked list with a tail pointer takes:

A. constant time

B. linear time

C. log time

D. *n* log *n* time

8. Removing the last item from a singly-linked list without a tail pointer takes:

A. *n* log *n* time

B. log time

C. linear time

D. constant time

9. Removing the first item from a singly-linked list with a tail pointer takes:

A. *n* log *n* time

B. linear time

C. log time

D. constant time

10. In a singly-linked list, you can move the tail pointer back one node in:

A. *n* log *n* time

B. linear time

C. constant time

D. log time

11. Suppose you have a pointer to a node in the middle of a singly-linked list. You can then delete that node in:

A. log time

B. *n* log *n* time

C. constant time

D. linear time

Concept: *doubly-linked lists (insertions)*

12. Appending to a non-circular, doubly-linked list without a tail pointer takes:

A. constant time

B. linear time

C. *n* log *n* time

D. log time

13. Appending to a non-circular, doubly-linked list with a tail pointer takes:

A. linear time

B. log time

C. n log n time

D. constant time

14. Removing the first item from a non-circular, doubly-linked list without a tail pointer takes:

    A. constant time

    B. linear time

    C. log time

    D. n log n time

15. Suppose you have a pointer to a node in the middle of a doubly-linked list. You can then insert a new node just after in:

    A. n log n time

    B. constant time

    C. linear time

    D. log time

16. Suppose you have a pointer to a node in the middle of a doubly-linked list. You can then insert a new node just prior with as few pointer assignments as:

    A. 1

    B. 2

    C. 5

    D. 4

    E. 3

17. **T** : **F**: Making a doubly-linked list circular removes the need for a separate tail pointer.

Concept: *doubly-linked lists (deletions)*

18. Removing the first item from a doubly-linked list with a tail pointer takes:

    A. constant time

    B. linear time

    C. n log n time

    D. log time

19. In a doubly-linked list, you can move the tail pointer back one node in:

    A. constant time

    B. linear time

    C. n log n time

    D. log time

20. In a doubly-linked list, what does a tail-pointer gain you?

    A. the ability to append the list in constant time

    B. the ability to remove the last element of list in constant time

    C. the ability to both prepend and remove the first element of list in constant time

    D. the ability to prepend the list in constant time

    E. the ability to remove the first element of list in constant time

F. the ability to both append and remove the last element of list in constant time