1. A set of data and operations to be performed on that data. Example: A queue.

2. The hidden bit is the highest order bit of the mantissa. This always has the value 1 and goes in the ones position.

3. A Last-In First-Out data structure guarantees that given a LIFO data structure containing \((n - 1)\) elements, when an \(n\)th element is inserted, that element must be removed before any of the previous \((n - 1)\) are eligible for removal. Example: A stack.

4. The length of the longest path in a tree, or the depth of the deepest node in a tree. Example: 7.

5. Big-O is useful when comparing the asymptotic running times of two different algorithms. In comparing Bubble Sort and Merge Sort, Big-O is useful.

6. Big-O is not useful when comparing two algorithms which have the same asymptotic running times. In comparing Bubble Sort and Quick Sort, Big-O is not useful.

7. (a)

\[
\lceil \log_2 26 + 10 + 1 + 1 + 1 \rceil = \lceil \log_2 39 \rceil \\
= 6 \text{ bits}
\]

(b)

\[
\lceil \log_2 180 \rceil = 8 \text{ bits}
\]

(c) If we want to do bit shifting, we can simply multiply and add: \(8 \times 6 + 8 = 54\) bits, which corresponds with 7 bytes. Without bit shifting, we need \(1 \times 8 + 1 = 9\) bytes.

8. (a) \(x / d \times + a g - b e c\)

(b) \(d a g + b e - x / c \times\)

9. (a) \(O(n)\)

(b) \(O(n)\)

(c) \(O(n^3)\)

(d) \(O(n^2)\)

(e) \(O(n^2)\)

10. (a) 205

(b) \(b7_{16}\)

(c) 376

11. (a) \(16622_8\)

(b) \(1d92_{16}\)

(c) \(11100010011011102\)
16. (a) Assume a pointer or index into the top of the stack, the array is static, and there is room. Push by incrementing the pointer and assigning to the referenced location. There is no worst case, or (for the pessimists) all cases are the worst case. The operation is $O(1)$.

(b) Implement with an in-order traversal. Again, no worst case. $O(n)$ where $n$ is the number of items in the tree.

(c) Traverse the list, comparing each element to the smallest element found thus far. This requires $O(n)$ time, where $n$ is the number of elements in the list. Again, there is no worst case.

17. It is an AVL-tree, because it is a binary tree, and for every node in the tree, the heights of the left and right sub-trees differ by at most 1.
20. (a) Positive
(b) $174 - 127 = 47$
(c) $1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{16} = 1.8125$
(d) $1.8125 \times 2^{17} = 2.55086697643 \times 10^{14}$

21. class Stack {
    public:
    Stack();
    void push(int value);
    int pop();
    int top();
    bool isEmpty();
private:
    Node *head;
}

Stack::Stack()
{
    head = 0;
}

void Stack::push(int value)
{
    Node *n = new Node(value);
    n->prev = 0;
    n->next = head;
    if (head)
        head->prev = n;
    head = n;
}

int Stack::pop()
{
    Node *n = head;
    int i;
    if (head) {
        head = head->next;
        head->prev = 0;
        i = n->val;
        delete n;
    }
return i;
}
error("pop");
}

int Stack::top()
{
    if (head)
        return head->val;
    error("top");
}

bool Stack::isEmpty()
{
    return !head;
}

22. void rotate(Stack *s, int n)
{
    Stack top, bottom;
    int i;

    if (n < 0)
        error("rotate");

    for (i = 0; !s->isEmpty(); i++)
        top.push(s->pop()); /* top has everything, */
        /* but backwards */
    if (i < n)
        error("rotate");

    for (i = n; i; i--)
        bottom.push(top.pop()); /* Put the bottom n in */
    while (!top.isEmpty()) /* bottom, forwards */
        s->push(top.pop()); /* Put the top back in */
    while (!bottom.isEmpty()) /* s, forwards */
        top.push(bottom.pop()); /* Reverse bottom, hold */
    while (!top.isEmpty()) /* it in top--sorry! */
        s->push(top.pop()); /* And put it in s, */
    } /* forwards again */

void roll(Stack *s, int n)
{
    Stack holder;
    int i, val;

    if (n < 0)
        error("rotate");

    for (i = 0; !s->isEmpty() && i != n; i++)
        holder.push(s->pop());

    if (i != n)
        error("rotate");
val = holder.pop();
while (!holder.isEmpty())
    s->push(holder.pop());
    s->push(val);