Problem 1 (16 points)

Determine the (worst-case) run-time requirements of the following algorithms. In each case, assume that the data-structure that is subject to the operation has \( n \) values stored in it. Justify your answer using a short and concise sentence.

1. A Deque (Double-Ended Queue):
   (a) push_front
   (b) push_back
   (c) pop_front
   (d) pop_back

2. An array-based list of unordered values:
   (a) insert
   (b) delete
   (c) lookup

3. Verify whether two arrays, each of which has \( n \) elements and the elements are sorted, contain the same values.

4. Verify whether two arrays, each of which has \( n \) elements and the elements are not sorted, contain the same values.
Problem 2 (20 points)

Using the following primitive functions:

- list null();
- int is_null(list p);
- int atom(list p);
- int eq(list p, list q);
- list car(list p);
- list cdr(list p);
- list cons(list p, list q);

and recursion, write a function to add two recursive lists.

Values are assigned to recursive lists based on these rules: () represents 1, ((())) represents 2, (((()))) represents 3, etc.

The function that you will write takes two such lists and constructs a list that represents the sum of the two argument lists. Each of the argument lists contains no atoms and at most one element, which may be deeply nested.

list sum(list p, list q)
// builds a list whose value is the sum of values of p and q.
// Examples:
// 1. sum of () and () is ()
// 2. sum of ((())) and (((()))) is (((((((()))))))). This is because the
//    value of the first list is 3 and the second list is 4. The sum
//    is a list that has 7 layers.
Problem 3 (20 points)

Your task is to write a function that given two lists, determines whether the first is a sub-list of the second.

```c
bool isASublist( list p, list q )
// Neither p nor q are atoms. Is p a sublist of q or not?
```

The function should return true given the lists (a) and ( (( (a) (b) (c) ) ) ) because the first list, (a), appears as a sub-list of the second list. In this example, in the second list, the first list is in bold-face to show the concept.

As another example,

```plaintext
((a) ((b))) is a sub-list of ((() a (b (c (a) ((b))) ))) ((e)))
```

Again, the first list has been emphasized in the second one to demonstrate the concept. For this problem, in addition to the list function that have been enumerated in the previous problem, you can use `equal`. Recall the definition of `equal` from Project 2:

```c
int equal(list p, list q);
```

`equal` takes two arbitrary recursive lists and determines if they are identical; that is, the parentheses are all in the same place and the atoms agree as to position and name.
Problem 4 (20 points)

Use a bit vector to determine whether an array of 9 integers contains all of the values 1 through 9. Here is the function prototype.

```c
bool hasDigits1_9(int a[])
```

// Array "a" has 9 elements and each element contains a value between 1
// and 9, inclusive. This function uses a bit vector to determine if "a"
// contains every integer exactly once or not. For example, if "a"
// contains values

2 6 4 3 9 1 8 7 2 5

// your function should return true, as this list contains each of the integers
// 1 through 9. On the other hand, for the following list, your function should
// return false.

2 6 4 2 9 1 8 7 2 5

// This is because the list doesn’t contain the integer 3. Of course, since
// the array has exactly 9 elements, then at least one of the other integers
// (2 in this case) must have been repeated.
// To solve this problem, you should use a bit vector (a single int or long)
// to represent the values in the array and then determine if all values are
// present. Specifically, you may not define new arrays.
Problem 5 (20 points)

For this problem, you are to write a function that reads and creates a list of Artists, the entity that you used in your first project. The definition of Artist appears on the last page of this exam.

```cpp
void function readArtists(std::fstream &stream, std::vector<Artist> &artists)
// "stream" is a file-stream that we have opened just before calling
// this function. The file that "stream" references contains a JSON
// array of Artist JSON Objects, similar to artists.json that you used
// in your first project. The structure of the input is:
// [ {attrs for artist 1}, {attrs for artist 2}, ..., {attrs for artist n}]

// This function reads and stores the artist JSON objects in vector, "artists".
// To that end, you can use the following function of Artist.

// void parseFromJSONstream(std::fstream &stream)

// parseFromJSONstream, when called, reads exactly one of the Artist objects.
// For example, {attrs for artist 1}. It expects the first non-space
// character to be a curley brace '{' and it reads the entire Artist
// JSON object; from the open-brace to the close-brace, inclusive.
// When necessary, here is how you can create and read an Artist JSON Object.

// Artist artist;
// artist.parseFromJSONstream(stream)

// This function doesn't return anything. It parses a JSON Array that
// consists of JSON Objects, each of which represents an artist, and
// stores them in "artists".
```