Exam 1

Select a minimal set of operations that distinguish `fsu::Vector` from other fsu:: sequential containers.

(Here **`i`** is an iterator, **`t`** is a container element, and **`n`** is an integer.)

`SetSize(n)`

Declare an **`fsu::Stack`** object **`S`** with elements of type **`fsu::String`** and underlying container**`fsu::Vector`**.

```
fsu::Stack < fsu::String , fsu::Vector < fsu::String > > S;
```

Which of the following are ***optional*** (but very desirable) components of an algorithm?

Run times

Select the answer that best describes the asymptotic runtime of the operation

`fsu::List::PushBack(t)`

Here **`t`** is an item of type **`fsu::List::ValueType`**and *n* is the size of the list.

O(1)

Select the most appropriate statement of the asymptotic runtime for the **`sequential_search`**algorithm with input size *n*.

O(*n*)

Select all choices such that

$2x^3 \log x + 3x^3 + 4x^2 + 5x - 100 <= O(f(x))$

$x^3 \log$ f(x) = $2x^3 \log x$ f(x) = $x^3 \log x + 1$ f(x) = $x^3 \log x + x + 1$

Declare an **`fsu::List`** object **`L`** with elements of type **`fsu::String`**.
`fsu::List<fsu::String> L;`

You are given the declaration

```
fsu::List<fsu::String> L;
```

Write a client program fragment that inserts the strings "Help", "I", "May", "You" into **`L`** (***in this order***) so that

```
L.Display(std::cout, ' ');
```

results in the sentence "`May I Help You`".

```
L.PushBack("Help");
L.PushFront("I");
L.PushFront("May");
L.PushBack("You");
```

The following represents output from the call d**`.Dump()`** from the **`fsu::Deque<char>`** object **`d`**:

```
content_[i]:  A B C D E F G H I J
   i mod 10:  0 1 2 3 4 5 6 7 8 9
                  e       b
```

What is the result of the **`d.Dump()`** ***after*** the call **`d.PopBack()`** ?

The following illustrates the contents of the control queue **`Q`** during a breadth-first search of the tree **`tree3`**, beginning at vertex **`R = root`** and searching for the vertex **`X = goal`**. (Search left before right, beginning at the root, as conventional for trees. The front of the queue is to the left.)

```
Q <--                                   (R = root)
------------                          /           \
R                                   (A)             (B)
...                               /     \         /     \
...                             (C)       (D)  (X = goal)  (E)
...                                     /    \
...                                   (F)      (G)
...
...                                       [tree3]
```

```
Q <--
------------
R
AB
BCD
CDXE
DXE
XEFG
```

This is an illustration of a vector of characters to be used in answering the question:

```
 element:   B  D  F  F  F  H  K  Q  W  Y
   index:   0  1  2  3  4  5  6  7  8  9
```

What is the return value of the call **`lower_bound('Z')`** ?

The following represents data of type char stored in a vector:

BDGGHJJK

With the initial search range for **`lower_bound`**underscored. Which if the following represents the search range after ONE iteration of the loop body in **`lower_bound(H)`** ?

BDGGHJJK

To copy the elements from an `fsu::Deque<> d` to an `fsu::Vector<> v` we can use the following generic algorithm call:

```
fsu::g_copy (d.Begin(), d.End(), v.Begin());
```

Write a traversal loop for an **`fsu::Vector`** object `v` .

```
for (size_t i = 0; i < v.Size(); ++i) { /* whatever */ }
```

Write the header of a loop that traverses the list x. Assume the declarations:

fsu::List < char > x;

fsu::ListIterator < char > i;

for (i = x.Begin(); i != x.End(); ++i)

EXAM 2

1. The adaptor template defining **`fsu::Stack < T , C >`** defines an implementation of **ADT Stack** by re-defining the user interface of the container **`C`**. Which of the operations listed below must **`C`** possess for this adaptation to compile? (Check all that apply.)

ANS: `PopBack()` `PushBack(t)`

2. Which of the following are ***optional*** (but very desirable) components of an algorithm?

ANS: **RunTime Proof:** THEOREM. *If the assumptions hold then the asserted runtime estimates are correct.*
RunSpace Estimate: Statement of the algorithm runspace requirements, in asympotic "+" notation, as a function of input size
RunTime Estimate: Statement of the runtime of the algorithm body, in asymptotic notation, as a function of input size
RunSpace Proof: THEOREM. *If the assumptions hold then the asserted runspace estimates are correct.*

3. Select the most appropriate statement for the asymptotic runtime for the **`binary_search`**algorithm with input size *n*.

ANS: $\Theta(\log n$

4. Suppose t is a balanced BST with *n* nodes. What is the worst-case runtime of t.Includes(x)? (Select the *best* answer.)

ANS: $\Theta(\log n)$

5. Select all choices that satisfy $22x^2 - 18x \log x + 15x - 9 <= O(f(x))$

ANS: $f(x) = 22x^2$ $f(x) = 22x^2$ $f(x) = 22x^2 + 15x - 9$

$f(x) = 22x^2 - 18x \log x$ $f(x) = x^2$ $f(x) = 18x^2 \log x$ $f(x) = x^2 \log x$

6. The code fragment below is intended to spell "BARK" on the screen after inserting the letters A, B, K, R (in this order) into the list L. There are lines of missing code:

```
fsu::List L;

fsu::List::Iterator i;

L.PushBack('A');

L.PushFront(('B');
```

```
L.PushBack('K');
i = L.Begin();

// ...

// missing lines

// ...

for (i = L.Begin(); i != L.End(); ++i)

  std::cout << *i;  // spells "BARK" on screen
```

Select the most appropriate lines of code for the missing lines.

```
ANS: ++i;

++i;

L.Insert(i,'R');
```

7. The following represents output from the call d.**`Dump()`** from the **`fsu::Deque<char>`** object **`d`**:

```
 content_[i]:  A B C D E F G H I J

    i mod 10:  0 1 2 3 4 5 6 7 8 9

                 e         b
```

What is the result of the following output statement?

```
std::cout << d;
```

```
G H I J A
```

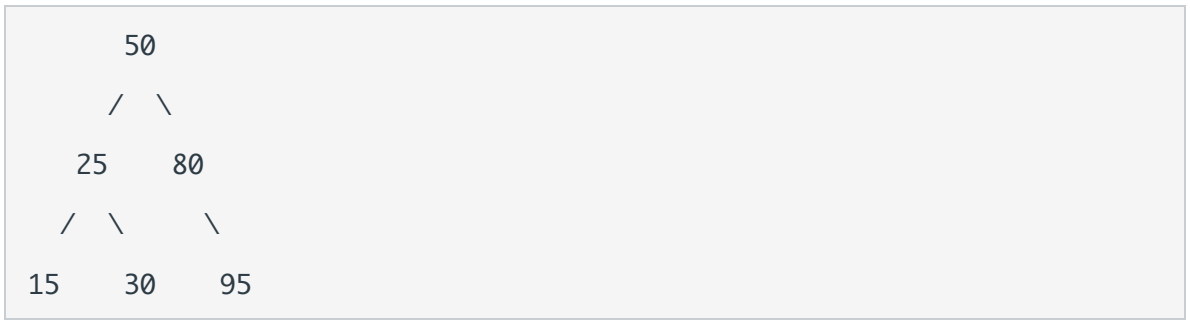
8. The following illustrates the contents of the control stack `S` during a depth-first search of a tree [`tree3`], beginning at vertex `R = root` and searching for the vertex `X = goal`. (Search left before right, beginning at the root.) The search process is started but not complete.

```
S -->                                    (R = root)
-----                                   /          \
R                                     (A)            (B)
RA                                   /    \         /    \
RAC                                (C)      (D)  (X = goal)  (E)
RA                                        /   \
RAD                                     (F)     (G)
RADF
...                                          [tree3]
```

Show the rest of the control stack as the algorithm completes the search.

```
S -->
-----
...
RAD
RADG
RAD
RA
R
RB
RBX
```

What is the result of a level-order traversal of the following tree?



ANS: 15 30 25 95 80 50

The following represents data of type char stored in a vector:

BDGGHJJK

With the initial search range for **`lower_bound`**underscored. Which if the following represents the search range after ONE iteration of the loop body in **`lower_bound(H)`** ?

BDGGHJJK

This is an illustration of a vector of characters to be used in answering the question:

```
element:   B  D  F  F  F  H  K  Q  W  Y
  index:   0  1  2  3  4  5  6  7  8  9
```

What is the return value of the call **`upper_bound('F')`** ?

5

To find a pointer `max` to the largest element in the array `a` we can use the following generic algorithm call (there are `size` elements in `a`)

```
max = fsu::g_max_element (a, a + size);
```

Given the code:

```
typedef KeyType                          String;
typedef DataType                         int;
AssociativeArray < KeyType, DataType > aa;
std::cout << aa["xyz"]; // (1)
aa["xyz"] = 20;         // (2)
std::cout << aa["xyz"]; // (3)
```

The key "xyz" is inserted into the table at line (1).

True

A function to display the contents of an associative array coule be implemented efficiently as follows:

```
Display(const AssociativeArray& aa)
  {
    KeyType key;
    for (key = aa.BeginKey(); key != aa.EndKey(); ++key)
      std::cout << key << " : " << aa[key] << '\n';
  }
```

False

Given the code fragment:

```
typedef KeyType                          String;
typedef DataType                         int;
AssociativeArray < KeyType, DataType > aa;
aa["xxx"] = 1;
aa["yyy"] = 2;
aa["xxx"] = 3;
Display(aa);
```

The result to screen should be

```
xxx : 1
yyy : 2
xxx : 3
```

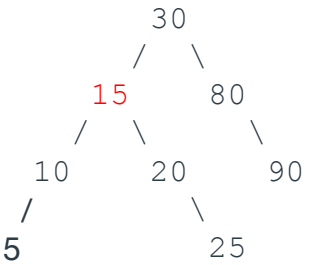
False

Suppose that `table` is a container that implements the Table API.

If `kval` is a key in the table, then `table.Retrieve(kval,dval)` overwrites the data associated with `kval` with `dval`.

False

The following tree:



Is a legal...

Red black tree

Define an *edge move* to be a call to any of the following `fsu::BinaryTree<>::Navigator`operations: `Initialize()` (i.e ., assignment to root), `++()`, `++(int)`, `--()`, or `--(int)`.

What, for a general binary tree, should the sum of all edge moves in an inorder traversal be? (*n* = size)

2n

The worst case run time [WCRT] for `BinaryTree<>::Iterator::operator++()` (where *n* = size of the tree) is:

Θ(*n*)

Amortized Θ(1)

This question is about internal operation/implementation of **`fsu::HashTable<K,D,H>`**. Suppose you have a hash table **`myTable`** as follows: **`KeyType`** is **`K = fsu::String`**, **`DataType`** is **`D = int`**, and **`HashType`** is **`H = hashclass::Simple<fsu::String>`**. The implementation of **`BucketType`** is **`fsu::List < fsu::Entry <K,D> >`**, it uses the **`BucketType::PushBack(entry)`** for table insert, and there are five (5) buckets. Recall that the **`Simple`** hash function just adds the character values of the string, starting with **`a = 0`**. Here is the code for the hash function:

```
uint64_t Simple (const char* S, size_t length)
{
  uint64_t rval(0);
  for (size_t i = 0; i < length; ++i)
  {
    rval += S[i] - 'a';
  }
  return rval;
}
```

Here is a table of hash values and for the 1-character keys:

```
key           a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s
t  u  v  w  x  y  z

hash_value    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25
```

What is the hash value and bucket number for the key **bcdb**?

This question is about internal operation/implementation of fsu::HashTable<K,D,H>. Suppose you have the following:

```
KeyType  is K = fsu::String

DataType is D = int

HashType is H = hashclass::Simple<fsu::String>
```

The implementation of BucketType is fsu::List<fsu::Entry<K,D>> and uses BuctetType::PushBack() for Table insert and there are five (5) buckets. Recall that the simple hash function just sums the character values in the string, offset so that Simple('a') = 0. Here is a table of has values of 1-character strings:

```
key           a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s
t  u  v  w  x  y  z

hash_value    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25
```

Assume that myTable is a hash table object of the type defined above and that the following code is executed:

```
myTable.Clear();

myTable.Insert("abc",1);

myTable.Insert("bbc",1);

myTable.Insert("bdg",1);

myTable.Insert("def",1);

myTable.Insert("xyz",1);

myTable.Dump(std::cout);
```

What is the result of the call to myTable.Dump() ?

```
b[0]: bdg:1
b[1]:
b[2]: def:1 xyz:1
b[3]: abc:1
b[4]: bbc:1
```

Suppose you have an associative array `aa` with key_type = `size_t`, and your code partner proposes the following loop to output the contents of `aa`:

```
for (size_t i = 0; i < max_size_t; ++i) { std::cout << i << ':' << aa[i] << '\n'; }
```

Are there any negative effects? Explain in complete sentences. Make a counter proposal if you have a better solution.

There are 2^64 potential keys in size_t - more than the number of atoms in the known universe. Most of these will be unused, but the loop will display them anyway ... Plus, because the AA bracket operator behaves as Insert(key, Data()) when key is not already in the table, after the loop all of the keys in size_t will now be in the aa, inflating it immensely, possibly running out of memory. The only solution is to use an Iterator-based standard traversal.