FINAL EXAM A

SECOND SEMESTER OF ACADEMIC YEAR 2022 – 2023

SOLUTION & SCORING CRITERION

1. LinkedLists (17pts)

```
bool separable(ListNode* list)
{
   int total_sum = 0;
   ListNode* tmp = list;
   while (tmp != nullptr)                    // 5pts
   {
      total_sum += tmp -> data;
      tmp = tmp -> next;
   }

   if (total_sum % 3 != 0)                   // 2pts
   {
      return false;
   }
   else
   {
      int sub_sum = total_sum / 3;           // 1pt

      int segement_count = 0;                // 2pts
      int current_sum = 0;

      tmp = list;
      while (tmp != nullptr)
      {
         current_sum += tmp -> data;
         if (current_sum == sub_sum)         // 5pts
         {
            segement_count++;
            current_sum = 0;
         }
         tmp = tmp -> next;
      }
      return segement_count == 3;            // 2pts
   }
}
```

2. String & sorting (15pts)

```
//函数 wordAccount 依次扫描 line 中单词，
// 调用 Captalize 将其转换为标准格式
// 将不同单词的个数记录在映射 wordAcc 中。
void wordAccount(string line, Map <string, int> &wordAcc)
{
    string word;

    int start = -1;                                          // 2pts
    for (int i = 0; i < line.length(); i++) {                // 1pt
       char ch = line[i];                                    // 1pt
       if (isalpha(ch)) {                                    // 2pts
         if (start == -1)                                    // 1pt
            start = i;                                       // 1pt
       } else {                                              // 1pt
          if (start >= 0) {                                  // 2pts
            word=Capitalize(line.substr(start, i - start));  // 2pts
            wordAcc[word]++;                                 // 1pt
            start = -1;                                      // 1pts
          }
       }
    }
}
```


3. Big-O (18pts)

a) O(N) // 6pts

b) O(logN) // 6pts

c) O(1) // 6pts

4. ADT-1 & Recursion (15pts)

```
bool subsetSumRec(const Vector<int> & a, int pos, Vector<bool> & used, int W){
   if (pos >= a.size())                                    // Base case: 6 pts
    {
        int sum = 0;
        for (int i = 0; i < a.size(); i++)
        {
          if (used[i])
         sum += a[i];
        }
        return (sum == W);
    }

    used[pos] = false;
    if (subsetSumRec(a, pos + 1, used, W)) return true;      // 2 pts
    used[pos] = true;
    if (subsetSumRec(a, pos + 1, used, W)) return true;      // 2 pts
    return false;                                            // 3 pts
}

bool subsetSum(const Vector<int> & a, int W)
{
     Vector<bool> used(a.size(), false);
     return subsetSumRec(a, 0, used, W);                     // 2 pts
}
```

5. ADT-2 & Recursion (17pts)

```
Set<Vector<string>> splitsRec(const string& str,const Vector<string>& chosen) {
      if (str == "") {                          // Base Case: 5pts
           return { chosen };
      }
      /* Otherwise, there are characters that need to get put into a piece of the
       * split. Try all ways of doing this.
       */
       else {                                   // 2 pts
           Set<Vector<string>> result;
           /* Munch off between 1 and all the characters, inclusive. */
           for (int i = 1; i <= str.length(); i++) {          // 4 pts
                string piece = str.substr(0, i);
                string remaining = str.substr(i);
```

```
            // Find all the splits we can make, assuming we commit to having this piece in front.
            result += splitsRec(remaining, chosen + piece);    // 2 pts
        }
      return result;                          // 2 pts
    }
}

Set<Vector<string>> splitsOf(const string& str) {
    return splitsRec(str, {});                // 2 pts
}
```

6. Trees & Recursion (18pts)

```
bool containsSubtree(Node * rootA, Node * rootB){
    if ((!rootA) || (!rootB))                 // Base case: 6 pts
      return (!rootA) && (!rootB);

    if (rootA->value == rootB->value && containsSubtree(rootA->left, rootB->left) &&    // 4 pts
            containsSubtree(rootA->right, rootB->right))
      return true;

    if (containsSubtree(rootA->left, rootB))      // 3 pts
      return true;

    if (containsSubtree(rootA->right, rootB))     // 3 pts
      return true;

    return false;                             // 2 pts
}
```