COMP 322: Fundamentals of Parallel Programming (Spring 2015)
Instructor: Vivek Sarkar
Worksheet 9: due at end of class today

Name: _____   **Netid: _____**

Parallelizing Pascal's Triangle with Futures and Memoization

There are four variants of the Binomial Cooefficients program provided in four different HJlib methods in the next page:

      a. Sequential Recursive without Memoization (`chooseRecursiveSeq()`)
      b. Parallel Recursive without Memoization (`chooseRecursivePar()`)
      c. Sequential Recursive with Memoization (`chooseMemoizedSeq()`)
      d. Parallel Recursive with Memoization (`chooseMemoizedPar()`)

Your task is to analyze the WORK, CPL, and Ideal Parallelism for these four versions, for the input N = 4, and K = 2. **Assume that each call to ComputeSum() has COST = 1, and all other operations are free.** Complete all entries in the table:

Variant	Work	CPL	Ideal Parallelism
`chooseRecursiveSeq`			
`chooseRecursivePar`			
`chooseMemoizedSeq`			
`chooseMemoizedPar`			

    Note: The work values should be equal for
    (i)   `chooseRecursiveSeq` and `chooseRecursivePar`
    (ii)   `chooseMemoizedSeq` and `chooseMemoizedPar`

Do you agree with the following statement: "Inefficient parallel programs often have more ideal parallelism than efficient parallel programs" in the context of this worksheet?

```java
1  private static int chooseRecursiveSeq(final int N, final int K) {
2      if (N == 0 || K == 0 || N == K) return computeBaseCaseResult();
3      final int left = chooseRecursiveSeq(N - 1, K - 1);
4      final int right = chooseRecursiveSeq(N - 1, K);
5      return computeSum(left, right);
6  }
7
8  private static int chooseRecursivePar(final int N, final int K) {
9      if (N == 0 || K == 0 || N == K) return computeBaseCaseResult();
10     final HjFuture<Integer> left = future(() -> chooseRecursivePar(N - 1, K - 1));
11     final HjFuture<Integer> right = future(() -> chooseRecursivePar(N - 1, K));
12     final HjFuture<Integer> resultFuture = future(() -> {
13         final Integer leftValue = left.get();
14         final Integer rightValue = right.get();
15         return computeSum(leftValue, rightValue);
16     });
17     return resultFuture.get();
18 }
19
20 private static final Map<Pair<Integer, Integer>, Integer> chooseMemoizedSeqCache = new ConcurrentHashMap<>();
21
22 private static int chooseMemoizedSeq(final int N, final int K) {
23     final Pair<Integer, Integer> key = Pair.factory(N, K);
24     if (chooseMemoizedSeqCache.containsKey(key)) {
25         final Integer result = chooseMemoizedSeqCache.get(key);
26         return result;
27     }
28     if (N == 0 || K == 0 || N == K) {
29         final Integer result = computeBaseCaseResult();
30         chooseMemoizedSeqCache.put(key, result);
31         return result;
32     }
33     final int left = chooseMemoizedSeq(N - 1, K - 1);
34     final int right = chooseMemoizedSeq(N - 1, K);
35     final int result = computeSum(left, right);
36     chooseMemoizedSeqCache.put(key, result);
37     return result;
38 }
39
40 private static final Map<Pair<Integer, Integer>, HjFuture<Integer>> chooseMemoizedParCache = new ConcurrentHashMap<>(
41 private static int chooseMemoizedPar(final int N, final int K) {
42     final Pair<Integer, Integer> key = Pair.factory(N, K);
43     if (chooseMemoizedParCache.containsKey(key)) {
44         final HjFuture<Integer> result = chooseMemoizedParCache.get(key);
45         return result.get();
46     }
47     final HjFuture<Integer> resultFuture = future(() -> {
48         if (N == 0 || K == 0 || N == K) {
49             return computeBaseCaseResult();
50         }
51
52         final HjFuture<Integer> left = future(() -> chooseMemoizedPar(N - 1, K - 1));
53         final HjFuture<Integer> right = future(() -> chooseMemoizedPar(N - 1, K));
54
55         final Integer leftValue = left.get();
56         final Integer rightValue = right.get();
57         return computeSum(leftValue, rightValue);
58     });
59     chooseMemoizedParCache.put(key, resultFuture);
60     return resultFuture.get();
61 }
```