CS101 Algorithms and Data Structures
Fall 2022
Homework 2

Due date: 23:59, September 25, 2022

1. Please write your solutions in English.
2. Submit your solutions to gradescope.com.
3. Set your FULL name to your Chinese name and your STUDENT ID correctly in Account Settings.
4. If you want to submit a handwritten version, scan it clearly. `CamScanner` is recommended.
5. When submitting, match your solutions to the problems correctly.
6. No late submission will be accepted.
7. Violations to any of the above may result in zero points.

1. (14 points) Multiple Choices

Each question has **one or more** correct answer(s). Select all the correct answer(s). For each question, you will get 0 points if you select one or more wrong answers, but you will get 1 point if you select a non-empty subset of the correct answers.

Write your answers in the following table.

(a)	(b)	(c)	(d)	(e)	(f)	(g)

(a) (2') Which of the following scenarios are appropriate for using hash tables?

A. The bookmark in the browser to store your favourite website URLs and their addresses.

B. An output stream buffer used to store the output data before stream flush to optimize write speed.

C. A debugger to trace back function calls of the program.

D. An index for the library management application to record the location of the books by their names.

(b) (2') Which of the following statements about the hash table are true?

A. We have a hash table of size $2n$ with a uniformly distributed hash function. If we store n elements into the hash table, then with a very high probability, there will be **no** hash collision.

B. In a hash table where collisions are resolved by chaining, an unsuccessful search (i.e. the required element does not exist in the table) takes $\Theta(1)$ on average if the load factor of the hash table is $O(1)$.

C. Lazy erasing means marking the entry/bin as erased rather than deleting it.

D. Rehashing is a technique used to resolve hash collisions.

(c) (2') Consider a table of capacity 11 using open addressing with hash function $h(k) = k \bmod 11$ and linear probing. After inserting 6 values into an empty hash table, the table is below. Which of the following choices give a possible order of the key insertion sequence?

Index	0	1	2	3	4	5	6	7	8	9	10
Keys			24	47	26	15	61	49			

A. 24, 47, 26, 15, 61, 49

B. 47, 24, 26, 49, 15, 61

C. 61, 24, 26, 47, 15, 49

D. 26, 61, 15, 49, 24, 47

(d) (2') Which of the following statements are true?

A. $f(n) = \Omega(g(n)) \implies g(n) = o(f(n))$.

B. $f(n) = O(g(n)) \implies g(n) = \omega(f(n))$.

C. $f(n) = \Theta(g(n)) \implies f(n) = o(g(n)) \wedge g(n) = \omega(f(n))$.

D. $f(n) = O(g(n)) \wedge f(n) = \Omega(g(n)) \implies f(n) = \Theta(g(n))$.

(e) (2') Consider the recurrence relation $a_n = (a_{n-1})^2$ where $a_1 = 2$. Which of the following statements are true?

A. $a_n = 2^{n-1}$
B. $a_n = O(n)$
C. $a_n = \Omega\left(3^n\right)$
D. $\log a_n = O(n)$

(f) (2') Suppose that an algorithm's running time is a function $f(n)$ of the input size n where $f(n) = g(n)2^n \log_2 n$ and $\frac{1}{2}n^2 \leq g(n) \leq n^2$. Which of the following statements are true?

A. The worst case time complexity is in $\Omega(n)$.
B. The best case time complexity is in $O(n^n)$.
C. The running time is $\omega(n^2 2^n \ln n)$.
D. The running time is $o(n^2 2^n \ln n)$.

(g) (2') Your new randomized algorithm for the max-clique problem runs in

$$T(n) = f(n)n^{1926} + 1$$

time, where n is the vertices number of the given graph and $0 \leq f(n) \leq 0.618$ is a random number depending on the input. Which of the following statements are true?

A. $T(n) = O\left(2^n\right)$.
B. $T(n) = \omega\left(n^{1025}\right)$.
C. The worst case running time is $o(n!)$.
D. The best case running time is $\Omega\left(\sqrt{\log \log n}\right)$.

2. (14 points) Hash table insertion simulation

Given a hash table with $M = 11$ slots and hash function $h_1(x) = (3x + 6) \bmod 11$. We want to insert integer keys $A = [33, 20, 2, 16, 23, 48, 35, 6, 31, 44]$.

(a) (4') **Chained hash table**

i. Suppose that collisions are resolved through chaining. If there exists a collision when inserting a key, the inserted key (collision) is stored at the end of a chain. Below is the hash table implemented by the array, and we want to insert all the keys into it. If a chain exists, please write the keys in the same order as they were inserted in the same cell and separate them by a comma.

Index	0	1	2	3	4	5	6	7	8	9	10
Keys											

Table 1: Chained hash table

ii. What is the load factor λ?

Solution:

(b) (5') **Linear probing hash table**

i. Suppose that collisions are resolved through linear probing. The integer key values listed below will be inserted in the order given. Write down the index of the home slot (the slot to which the key hashes before any probing) and the probe sequence (do not write the home slot again) for each key. If there's no probing, leave the cell blank.

Key Value	33	20	2	16	23	48	35	6	31	44
Home Slot										
Probe Sequence										

Table 2: Linear probing sequence

ii. Write down the content of the hash table after all the insertions.

Index	0	1	2	3	4	5	6	7	8	9	10
Keys											

Table 3: Linear probing hash table

(c) (5') **Double hashing hash table**

i. Suppose that collisions are resolved through double hashing. The probing function is described as

$$H_i(k) = (h_1(k) + i \cdot h_2(k)) \bmod 11$$

for any give key value k in the i-th probing (i starts from 0). $h_2(k)$ is the second hash function defined as

$$h_2(k) = 7 - (k \bmod 7)$$

Write down the index of the home slot and the probe sequence for each key.

Key Value	33	20	2	16	23	48	35	6	31	44
Home Slot										
Probe Sequence										

Table 4: Double hash sequence

ii. Write down the content of the hash table after all the insertions.

Index	0	1	2	3	4	5	6	7	8	9	10
Keys											

Table 5: Double hashing hash table

3. (10 points) Establishing the asymptotic bounds

You are trying to compute the coefficients of the polynomial $B_n(x) = (1+x)^n$

```
class Polynomial {
  std::vector<double> coeff;

 public:
  Polynomial(std::size_t deg) : coeff(deg + 1) {}
  std::size_t deg() const { return this->coeff.size() - 1; }
  const double &operator[](std::size_t i) const { return coeff[i]; }
  double &operator[](std::size_t i) { return coeff[i]; }
  Polynomial operator*(const Polynomial &rhs) const {
    Polynomial prod(deg() + rhs.deg());
    for (std::size_t i = 0; i <= deg(); i++)
      for (std::size_t j = 0; j <= rhs.deg(); j++)
        prod[i + j] += coeff[i] * rhs[j];
    return prod;
  }
};
```

In the following questions, you may assume that arithmetic operations of 64-bit floating-point numbers take *constant time*.

NOTE: Please clearly demonstrate your steps. The answer alone only accounts for 1pt in every sub-problem.

- Show the recurrence relation and explain its meaning,
- show your upper/lower bound estimator,
- and show your answers.

(a) (1') For any non-negative integer n, find out $\deg(B_n(x))$.

Solution:

(b) (1') What is the time complexity of the polynomial multiplication for the given implementation? Suppose the two operands are of degree n and m respectively.

Solution:

(c) (4') **First trial: brute force solution**

You decided to use an iterative algorithm: First find $B_1(x)$, and then $B_2(x)$, and so on. The solution can be implemented with the following C++ code.

```
Polynomial solve(std::size_t n) {
  if (n == 0) {
    Polynomial result(0);
    result[0] = 1;
    return result;
  }
  Polynomial factor(1);
  factor[0] = factor[1] = 1;
  return factor * solve(n - 1);
}
```

Please find the asymptotic tight bound of running time of this algorithm. Your answer should be like $\Theta(\cdot)$.

Solution:

(d) (4') **Accelerate the Algorithm with Doubling**

After STFW (Searching The Friendly Web), you come up with a doubling algorithm:

```
Polynomial solve(std::size_t n) {
  if (n == 0) {
    Polynomial result(0);
    result[0] = 1;
    return result;
  }
  auto half = solve(n / 2);
  auto double_half = half * half;
  if (n % 2 == 1) {
    Polynomial factor(1);
    factor[0] = factor[1] = 1;
    return factor * double_half;
  }
  return double_half;
}
```

Please find the asymptotic tight bound of running time of this algorithm. Your answer should be like $\Theta(\cdot)$.

Solution: