CS146 Fall 2024 – Final

Name: _____ **Section: A**

Date: _____ **Start time:** _____ **End time:** _____

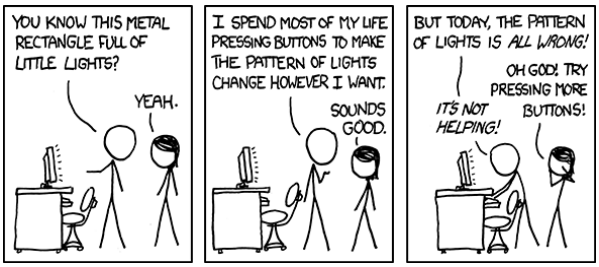
Honor Code:

Signature: _____

This exam is closed book, closed notes, closed computer, closed calculator, etc. You may only use (1) the final "cheat sheet" from the course page, and (2) a single double-sided letter sheet of notes of your own creation. **You have 3 hours.** Read the problem descriptions carefully and write your answers clearly and *legibly* in the space provided. Circle or otherwise indicate your answer if it might not be easily identified. You may use extra sheets of paper, stapled to your exam, if you need more room, as long as the problem number is clearly labeled and your name is on the paper. If you attached extra sheets indicate on your main exam paper to look for the extra sheets for that problem.

You do need to include module imports (if relevant for your code), but do not need to include comments, docstrings or constants in your code.

Complete problems 17-20 and any of the problems 1-16 for which you have not previously scored a 4 ("Exemplary"). Each problem is assessed in its entirety. Thus you should complete each problem fully, even if you previously answered some parts correctly.



Question 17: Searching and sorting algorithms

(a) The following functions are intended to return the index of `val` in a sorted list, or -1 if `val` is not found. All of the functions have the same docstring, shown below right. Each function implements one of the algorithms listed below left.

1. iterative linear search
2. recursive linear search
3. iterative binary search
4. recursive binary search

```
"""Search for val in sorted list lst

Args:
 lst: A sorted list.
 val: The value to search for in lst.
 lo: Lower index of search range, initially 0
 hi: High index of search range, initially len(lst)-1

Returns: Index of val or -1 if not found.
"""
```

Match each function to the algorithm it implements by writing the number on the line.

A. _____

```
def searchA(lst, val, lo, hi):
    if lo > hi:
        return -1
    m = (lo + hi) // 2
    if lst[m] == val:
        return m
    elif lst[m] > val:
        return searchA(lst, val, lo, m-1)
    else:
        return searchA(lst, val, m+1, hi)
```

B. _____

```
def searchB(lst, val, lo, hi):
    while lo <= hi:
        m = (lo + hi) // 2
        if lst[m] == val:
            return m
        elif lst[m] <= val:
            lo = m+1
        else:
            hi = m-1
    return -1
```

C. _____

```
def searchC(lst, val, lo, hi):
    if lo > hi:
        return -1
    elif lst[lo] == val:
        return lo
    else:
        return searchC(lst, val, lo+1, hi)
```

(b) For the remaining 4^{th} algorithm, provide the missing function definition below. Your function should be named `searchD` and be usable in place of any of the functions above. Write the corresponding algorithm number on the line below. Your solution will be exclusively evaluated on correctness.

D. _____

Question 18: Use big-O analysis to inform algorithm design

(a) Which of the following best describes the role of Big O analysis? Big O analysis:

- ◯ is used to compare the exact execution times of different algorithms.
- ◯ is limited to understanding an algorithm's execution time.
- ◯ is defined as the worst-case behavior of an algorithm.
- ◯ is used to compare how execution times change as the input size grows.

(b) Python can represent integers larger than the native size of the computer (generally 64 bits) by storing the individual digits as a list. We can assume these "bignums" never have more than 10 digits, thus, adding two bignums will require more than 1 operation but fewer than some constant c. In contrast, adding two 64-bit "regularnums" only requires 1 operation. Which of the following best describes the time complexity and actual execution time of summing n bignums or summing n regularnums?

- ◯ Summing regularnums has $\mathcal{O}(1)$ time complexity, while summing bignums is $\mathcal{O}(n)$ and will be slower in practice.
- ◯ Both sums have $\mathcal{O}(n)$ time complexity but summing regularnums will be faster in practice.
- ◯ Both sums have $\mathcal{O}(n)$ time complexity and the similar executions times in practice.
- ◯ Summing regularnums has $\mathcal{O}(n)$ time complexity, while summing bignums is $\mathcal{O}(n^2)$ and will be slower in practice.

Briefly explain your choice. Include in your explanation a calculation of $f(n)$ the number of operations required to sum n bignums and n regularnums. Recall for `sum` we are effectively executing the following code:

```
result = 0
for num in nums: # nums is a list of length n
    # '+' is 1 operation for regularnums, and > 1 but <= `c` operations for bignums
    result = result + num
```

Question 19: Understand numeric representations and associated operations

(a) Add the following binary numbers assuming they both are unsigned positive integers. Show your work (including all carries) and verify your answer by converting both numbers and their sum to decimal.

```
  0110111
+ 1000011
---------
```

(b) You wrote the following code to add two positive numbers encoded with an arbitrary base (greater than 1). The numbers are stored as lists of integer digits, with the least significant digit in index 0. For example, the number 123 in base 10 would be stored as `[3, 2, 1]`. You can assume both numbers have the same number of digits (the lists have the same length). Unfortunately, some portions of the code were accidentally deleted. Identify which of the code snippets below should replace the missing portions of the code, e.g., `?? Missing A ??`, by writing the letter on the line. Not all code blocks are needed. One answer is provided as an example. Some example usage is shown to the right.

```
def add(base, num1, num2):
    """Return num1 + num2 for numbers encoded in base."""
    result = []
    carry = 0
    for i in range(len(num1)):
        next_digit = ?? Missing A ??
        if ?? Missing B ??:
            next_digit = next_digit - base
            carry = 1
        else:
            carry = 0
        result.append(next_digit)
    if ?? Missing C ??:
        result.append(1)
    return result
```

```
# 4 + 6 in base 10 and base 2
>>> add(10, [4], [6])
[0, 1]
>>> add(2, [0, 0, 1], [0, 1, 1])
[0, 1, 0, 1]
```

i. _____**B**_____ `next_digit >= base`

ii. _____ `num1[i] + num2[i]`

iii. _____ `num1[i] + num2[i] + carry`

iv. _____ `carry != 0`

v. _____ `carry == 0`

vi. _____ `next_digit != 0`

Question 20: Vectorized execution

Translate each of the code blocks below to just use Python built-in functions and operators, assuming that `x` and `y` are *non-empty* lists of floats of the same length (instead of a NumPy vector). If the function returns a vector, your function should return a list. The functions `min`, `max`, `sum` and the `math` module are considered built-in Python. Your solution will be evaluated based on the correctness and efficiency of your approach.

(a)
```
def mystery(x, y):
    return np.sum(y[x == y])
```

(b)
```
def mystery(x, y):
    return (y + 3) / x
```