Assignment 1

CS Osvita

January 6, 2025

1 API

Calculate the number of times an API is called as a function of input size n. Provide a detailed explanation of how the input size influences the number of API calls. Understanding this relationship is crucial for optimizing application performance. Note: in all examples, integer division is used.

```
a.  for(int i = 5; i < n/2; i += 5)
        api()

b.  for (int i = n; i > 1; i /= 2)
        api()

c.  for (int i = 0; i * i < n; i++)
        api()

d.  for (int i = 0; i < n; i++)
        for (int j = 0; j < 100; j++)
            api()

e.  for (int i = 1; i <= n * n; i++)
        for (int j = 1; j <= i; j++)
            api()

f.  for (int i = 1; i <= n; i++)
        for (int j = 1; j < i; j *= 2)
            api()

g.  for (int i = 1; i <= n; i++)
        for (int j = 1; j <= i; j++)
            for (int k = 1; k <= j; k++)
                api()
```

```
    // Hint: harmonic number
h.  for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j += i)
            api()
        for (int j = 1; j <= i; j++)
            api()

i.  void f(int n)
        if n == 0: return
        for (int i = 0; i < n; i++)
            api()
        f(n/2)

j.  void f(int n)
        if n == 0: return
        for (int i = 0; i < n; i++)
            op()
        f(n/2)
        f(n/2)

// Estimate the lower bound of the time complexity
k.  void permute(prefix: array, n: int)
        if prefix.length == n:
            print(prefix)
            return
        for (int num = 0; num < n; num++)
            if not prefix.contains(num):
                permute(prefix + [num], n)
    permute(prefix=[], n=3) // function call example
```

2 Memory

A. What is the space complexity of examples **i**, **j**, **k** in the first task? Note that no tail recursion is applied.

B. The text consists of N words. The length of each word is not greater than L. The text has a total of A different characters. To solve a certain problem, it is necessary to split the text into words and create a hash table from them so that it is possible to determine if a certain word is in the text quickly. How much memory is needed to store the dictionary? Why?

1. N **2.** N^2 **3.** $N * L$ **4.** $N * A$ **5.** $N * L * A$

3 Web Server

There is a web server designed to manage a TODO list. The list's items are stored in a Txt file on the disk. A new line separates each item. Evaluate the time complexity (using big O notation) of the following API methods, assuming there are N items stored in the file and the size of 1 item is no more than 280 characters. Consider the operations these methods perform and their implications on time complexity in your programming language. Explain your reasoning behind each time complexity calculation.

```python
@app.route('/', methods=['GET'])
def paginated_get():
    # Get the current page number from the request arguments (hash table lookup)
    page = int(request.args.get('page', '0'))
    first = page * PAGE_SIZE
    last = first + PAGE_SIZE
    result = []
    # Open the file in read mode
    with open(FILE_PATH, 'r') as f:
        # Iterate over the lines within the range of [first, last)
        for i, line in enumerate(f.readlines()[first:last]):
            # Append a dictionary containing the line id and data to the result array
            result.append({'id': first + i, 'data': line.strip()})
    # Return the result list as a JSON response
    return {"result": result}

@app.route('/', methods=['POST'])
def post():
    # Extract the 'data' field from the JSON payload in the request (hash table lookup)
    data = request.json['data']
    with open(FILE_PATH, 'a') as f:
        f.write('\n' + data)
    # Return an empty response with a status code of 201 (Created)
    return {}, 201

@app.route('/<int:data_id>', methods=['PUT'])
def put(data_id):
    data = request.json['data']
    with open(FILE_PATH, 'r') as f:
        # Read all lines from the file into a list
        new_data = f.readlines()
        # Update the specified 'data_id' line with 'data' and append a newline.
        new_data[data_id] = data + '\n'
    with open(FILE_PATH, 'w') as f:
        f.writelines(new_data)
    return {}, 204
```

```
@app.route('/<int:data_id>', methods=['DELETE'])
def delete(data_id):
    with open(FILE_PATH, 'r') as f:
        new_data = f.readlines()
    del new_data[data_id]
    with open(FILE_PATH, 'w') as f:
        f.writelines(new_data)
    return {}, 204
```

4 Function

Find the maximum number of consecutive identical characters in a string:

```
def max_consecutive_elements(input_str: str) -> int:
    result, cur_idx = 0, 0
    while cur_idx < len(input_str):
        next_idx = cur_idx
        while next_idx < len(input_str) and input_str[next_idx] == input_str[cur_idx]:
            next_idx += 1
        result = max(result, next_idx - cur_idx)
        cur_idx = next_idx
    return result
```

The code of the function *max_consecutive_elements* is complex to understand. It can be visually simplified by extracting a separate function that will search for the next different character.

```
def get_next_diff(input_str: str, cur_idx: int) -> int:
    next_idx, n = cur_idx, len(input_str)
    while next_idx < n and input_str[next_idx] == input_str[cur_idx]:
        next_idx += 1
    return next_idx

def max_consecutive_elements(input_str: str) -> int:
    result, cur_idx = 0, 0
    n = len(input_str)
    while cur_idx < n:
        next_idx = get_next_diff(input_str, cur_idx)
        result = max(result, next_idx - cur_idx)
        cur_idx = next_idx
    return result
```

Read the code of the function *get_next_diff*. How long does it take to work depending on the length of the string N in the worst case?

- 1 or N

And now look at the function *max_consecutive_elements*. How many times will it call the function *get_next_diff* depending on the length of the string N in the worst case?

- 1 or N

What is the time complexity of the function *max_consecutive_elements* depending on the length of the string N in the worst case?

- N or N^2

5 Estimations

1. A sorting algorithm takes 1 second to sort 1000 items on your local machine. How long will it take to sort 10000 items?
   (a) if the algorithm takes time proportional to n^2
   (b) if the algorithm takes time proportional to $nlogn$
2. How many instructions can your CPU execute in one year if the machine is always left running?
3. Consider the following algo performance: n^{100} vs 1.01^n. Which one would you choose and why?
4. Is $2^{n+1} =$ BigTheta(2^n)? Why? Provide a formal proof.

6 Coding

1. Solve the following problem using at least two different algorithms and send submission links.
2. After exploring different methods, review the first 12 minutes of presentation and implement the algorithm discussed. Send a submission link.
3. What are the worst-case, average-case and best-case time complexities of my solution? The average-case time complexity is a new concept for you. You should investigate and apply the concept of expected-value (more examples in Cormen, chapter 5: Probabilistic Analysis and Randomized Algorithms).

```python
def majority_element(arr: list) -> int:
    majority_cnt = len(arr) // 2
    while True:
        candidate = random.choice(arr) # Select a random item from the arr
        if compute_frequency(arr, candidate) > majority_cnt:
            return candidate

def compute_frequency(arr: list, candidate: int) -> int:
    freq = 0
    for el in arr:
        if el == candidate:
            freq += 1
    return freq
```