Experiment with ChatGPT

This is a mere experiment with ChatGPT to showcase how we can leverage the power of AI to boost our productivity. In this document I have shown how to brainstorm with ChatGPT and derive a solution from it.
Though the results are impressive, there are major issues that one may face if he or she decides to use AI models such as this without proper understanding of software development.

But in contrast, these actually can be used as significant productivity boosters for those who know what they are doing.

I am posting my initial analysis below that I did without the help of ChatGPT so that we can observe what ChatGPT does well and where it lacks.

There is one big issue that causes slowness in the code

- ***The 'grantVacation' function iterates through an array of payroll objects and in each iteration it goes into 2 inefficint operations to fetch address information and employee information by looping through arrays 'addresses' and 'employees'. It takes O(n) time to fetch them from those arrays in a given iteration of the outer loop.***

Hence the overall time complexity of 'grantVacation' function becomes O(n^2) as it has this nested loop situation.
If we use a different data structure other than arrays which we can use to fetch address and employee information in O(1) time, we can fix this slowness.
I have provided an alternative implementation for this in python language using dictionaries which has a O(1) lookup time.

I also spot two other minor issues in the implementation.

- ***The function 'yearsSince' gets called for each payroll inedx and it calculates millisecondsPerYear variable each time. It can either be replaced with its value inplace or the value can be obtained from a predefined global level constant.***

- ***Assuming we are sending the emails in a single day, we don't need to create a new Date() object in each iteration while looping the payroll***

There is a small implementation detail that I would like to highlight is as below
```
const yearsEmployed = yearsSince(empInfo.endDate, today);
```

The above line is as per the original code provided in the assignment
But I believe the line should be amended as below
```
const yearsEmployed = yearsSince(empInfo.startDate, today);
```

Q1.

```
    let newVacationBalance = yearsEmployed + payrollInfo.vacationDays;

    emailApi.queueEmail(
      emailBatchId,
      addressInfo.email,
      "Good news!",
      `Dear ${empInfo.name}\n` +
      `based on your ${yearsEmployed} years of employment, you have been granted
${yearsEmployed} days of vacation, bringing your total to ${newVacationBalance}`
    );
  }
  await emailApi.flushBatch(emailBatchId);
}
```

The provided code exports three interfaces: AddressBook, Payroll, and EmailApi. The AddressBook interface has properties for an employee's ID, first name, last name, and email address. The Payroll interface has properties for an employee's ID and number of vacation days. The EmailApi interface has methods for creating a batch of emails, queueing individual emails to that batch, and flushing the batch of emails to send them.

The code also includes a function called "yearsSince" that takes two dates as parameters and calculates the number of years between them. And then there is a function called "grantVacation" that takes in three arrays and an EmailApi object as parameters. This function is designed to grant bonus vacation days to employees based on their years of experience, and then send them an email notification of the bonus using the provided EmailApi. The function loops through the "payroll" array, and for each element, it finds the corresponding employee's address and personal details from the "addresses" and "employees" arrays. It then calculates the employee's years of experience using the "yearsSince" function and the employee's start and end date. And finally, it sends an email to the employee with their new vacation balance.

This is where I initially pasted the code and it correctly describes the functionality, which is very Impressive.

Q2.

grantVacation function is slow. why

The "grantVacation" function may be slow for a few reasons:

1. The function loops through the entire "payroll" array, and for each iteration, it performs a find operation on the "addresses" and "employees" arrays. This can be slow if the arrays are large, and the find operation is not optimized.
2. The "yearsSince" function uses the Date object to calculate the difference between two dates. This can be a computationally intensive operation, especially if the dates are far apart.
3. The "flushBatch" method of the EmailApi is asynchronous, which means that the function will wait for all the emails to be sent before moving on to the next iteration of the loop. This can add significant delay to the overall execution of the function.

To optimize this function, you could consider the following:

1. Use a data structure that allows for faster lookups, such as a hash table, instead of arrays.
2. Use a more efficient algorithm to calculate the difference between two dates, such as one that avoids the use of the Date object.
3. Use a more efficient email sending library or service that allows for sending multiple emails in parallel, instead of waiting for each email to be sent before moving on to the next one.

Here I ask the reason for the slowness by giving a pre-context to it. I say its slow and ask why from it.

It outputs 3 reasons.

- Reason 1 is correct.
- Reason 2 is not correct. I am not familiar with typescript but I can clearly understand that there is no need to do value iteration to derive the number of days. Hence the dates being far apart is not a concern here.
- Reason 3 can be partially true. But we have no reason to debug the email sending functionality. Besides, flush function is not called under a loop and ChatGPT has got that point wrong. The function simply sends out a batch in each call.

ChatGPT even outputs possible solutions as well and that is really helpful.

The bottom line here is that the user should be knowledgeable enough to filter out what's right and wrong.
But on the plus side, it draws the user's attention into different directions and it really helps at a level of brainstorming.

Q3.

what is the overall time complexity of grantVacation

The overall time complexity of the "grantVacation" function is O(n^2) where n is the number of elements in the payroll array.

The reason is that the function loops through the payroll array, and for each element, it performs a "find" operation on the addresses and employees arrays. The "find" operation has a time complexity of O(n), as it needs to iterate through the entire array to find the matching element.

Since the function performs the find operation twice for each element in the payroll array, the overall time complexity is O(n) * O(n) = O(n^2).

This complexity could be improved if we change the data structure of the address and employee arrays to use some data structure that allows faster lookups, such as a hash table. That would change the time complexity of the find operation to O(1) and the overall time complexity of the function to O(n) which is a significant improvement.

It is right about the time complexity. It also goes an extra step and suggests that we use hash tables instead of arrays to improve performance without even being asked. How impressive!!

Q4.

rewrite only grantVacation function in python with O(n) overall time complexity

Here's an example of how you could rewrite the "grantVacation" function in Python with an overall time complexity of O(n), assuming that the payroll, addresses, and employees data is stored in lists of dictionaries:

Copy code

```
from datetime import datetime

def years_since(start_date, end_date):
    return (end_date - start_date).days / 365.25

def grant_vacation(email_api, payroll, addresses, employees):
    today = datetime.now().date()
    address_lookup = {address['emp_id']: address for address in addresses}
    employee_lookup = {employee['id']: employee for employee in employees}
    email_batch_id = email_api.create_batch()
    for payroll_info in payroll:
        emp_id = payroll_info['emp_id']
        address_info = address_lookup[emp_id]
        emp_info = employee_lookup[emp_id]
        end_date = emp_info.get('endDate', today)
        years_employed = years_since(end_date, today)
        new_vacation_balance = years_employed + payroll_info['vacationDays']

        email_api.queue_email(
            email_batch_id,
            address_info['email'],
            "Good news!",
            f"Dear {emp_info['name']},\n"
            f"based on your {years_employed} years of employment, you have been granted
{years_employed} days of vacation, bringing your total to {new_vacation_balance}"
        )
    email_api.flush_batch(email_batch_id)
```

Now this is the tricky part. If a user tries to simply copy paste ChatGPT code, a lot of things can go wrong.

- Firstly it has completely forgotten the async-await usage that it originally had
- Secondly, it automatically assumes that we are providing an array of dictionary objects instead of python objects
  - What if we try to generate the whole code instead of the individual ‘grant_vacation’ function? Would it replace nested dictionaries with Python objects, as it has more context now than before? Actually yes. Please see the below example that I created to showcase just that.

```python
async def grantVacation(emailApi: EmailApi, payroll: list[Payroll], addresses:
list[AddressBook], employees: list[Employee]):
    email_batch_id = emailApi.createBatch()
    addresses_dict = {address.emp_id: address for address in addresses}
    employees_dict = {employee.id: employee for employee in employees}
    for payroll_info in payroll:
        address_info = addresses_dict[payroll_info.emp_id]
        emp_info = employees_dict[payroll_info.emp_id]

        today = datetime.datetime.now()
        years_employed = yearsSince(emp_info.startDate, today)
        new_vacation_balance = years_employed + payroll_info.vacationDays

        emailApi.queueEmail(
            email_batch_id,
            address_info.email,
            "Good news!",
            f"Dear {emp_info.name}\n" +
            f"based on your {years_employed} years of employment, you have been granted
{years_employed} days of vacation, bringing your total to {new_vacation_balance}"
        )
    await email
```

- It created classes and lists of objects properly. It even fetches properties from the objects properly to populate the dictionaries.
- But the generated function was a partially generated one. (check the await line)
- Also note that here it actually has included async-await as it sees more context overall

- On the plus side,
  - it has included a pre-calculated float literal value for year_since function

```python
def years_since(start_date, end_date):
    return (end_date - start_date).days / 365.25
```

  - It also has moved ‘today’ date creation out from the loop

```python
    today = datetime.now().date()
    address_lookup = {address['emp_id']: address for address in addresses}
```

Q5.

Additionally we can do things such as making the dictionaries to be passed as arguments

pass address_lookup and employee_lookup as arguments to grant_vacation function

Sure, here's an example of how you could pass the address_lookup and employee_lookup dictionaries as arguments to the "grant_vacation" function:

Copy code

```
def grant_vacation(email_api, payroll, address_lookup, employee_lookup):
    today = datetime.now().date()
    email_batch_id = email_api.create_batch()
    for payroll_info in payroll:
        emp_id = payroll_info['emp_id']
        address_info = address_lookup[emp_id]
        emp_info = employee_lookup[emp_id]
        end_date = emp_info.get('endDate', today)
        years_employed = years_since(end_date, today)
        new_vacation_balance = years_employed + payroll_info['vacationDays']

        email_api.queue_email(
            email_batch_id,
            address_info['email'],
            "Good news!",
            f"Dear {emp_info['name']},\n"
            f"based on your {years_employed} years of employment, you have been granted
{years_employed} days of vacation, bringing your total to {new_vacation_balance}"
        )
    email_api.flush_batch(email_batch_id)

# usage
address_lookup = {address['emp_id']: address for address in addresses}
employee_lookup = {employee['id']: employee for employee in employees}
grant_vacation(email_api, payroll, address_lookup, employee_lookup)
```

By passing the address_lookup and employee_lookup dictionaries as arguments, you can avoid the need to create them within the function. This allows you to create and populate the dictionaries once, outside of the function, and then reuse them for multiple calls to the function. This way the overall time complexity remains O(n)

Conclusion

As I have demonstrated above, it takes only a few minutes to generate insights about code and simple functionalities out of the box. This actually means that developers now could spend more time on solving the real business problems rather than worrying about trivial functionalities. I personally see this as a great productivity booster. One great use case would be unit test case generation.

Some developers fear the rise of AI as they think AI might eventually replace their jobs. But in reality AI will not replace them. But if they don’t use AI, someone using AI might replace them as those who use AI will be far more productive than a traditional developer. In fact, developers already use stack overflow to seek aid for the code that they write which is pretty much analogous to usage of AI for code generation (use with caution).

I also believe that organizations in general should be more aware and align themselves to be more productive with the help of AI. There should be proper frameworks to encourage and guide the developers to use AI efficiently in their daily tasks.

As developers, we should understand that these tools are not going away and they are here to stay. Organizations will eventually catch up with this and would start expecting more productivity from developers as some part of their burden is being taken over by AI. So, its very important that we as developers become familiar with these tools.

Finally the bottom line is that from the organizations side, they should think about how to align their goals and strategies and how to gain an edge with the help of AI. And from the developers side, they should correctly identify what are the important skills that they should focus on improving in order to thrive in the era of AI.