Lab 3 Report

Zhaoyi Wang 1689747

Part 1: Benchmarking in Rust

Question 1

```
.LCPI0_0:
        .long 10
        .long 0
        .long 12
        .long 32

example::main:
        sub     rsp, 16
        movaps  xmm0, xmmword ptr [rip + .LCPI0_0]
        movups  xmmword ptr [rsp], xmm0
        xor     eax, eax
        cmp     dword ptr [rsp + 4], 32
        setg    al
        mov     dword ptr [rsp + 8*rax + 4], 10
        add     rsp, 16
        ret
```

```
Data declarations:
        Declare 4 4-byte values, referred to as .LCPI0_0, initialized to
10, 0, 12, 32

Intro to main() function:
        Subtraction: stack pointer - 16
        Move Aligned Packed Single-precision: Send the memory content
value to the destination register (xmmword ptr [rip + .LCPI0_0] to xmm0)
        Move Unaligned Packed Single-precision: Send the memory content
value to the destination register (xmm0 to xmmword ptr [rsp])
        XOR (exclusive or): eax xor eax
        Compare: Compare (dword ptr [rsp + 4]) with 32
        Set if greater: set al to 1 if greater (if not, set 0)
        Move: Copy data 10 to [rsp + 8*rax + 4]
        Add: rsp + 16
        Return
```

Overall, this piece of code plans to compare a set of values. If 32 is greater than the value of the specified memory address, then 10 is copied to its destination address.

Question 2

```
fn selection_sort(arr: &mut [i32]) {
    let len = arr.len();
    // Rust would skip iteration if lower bound >= upper bound.
    // Hence, no need to `len - 1`.
    for i in 0..len {
        let mut temp = i;
        for j in (i + 1)..len {
            if arr[temp] > arr[j] {
                temp = j;
            }
        }
        arr.swap(i, temp);
    }
}
```

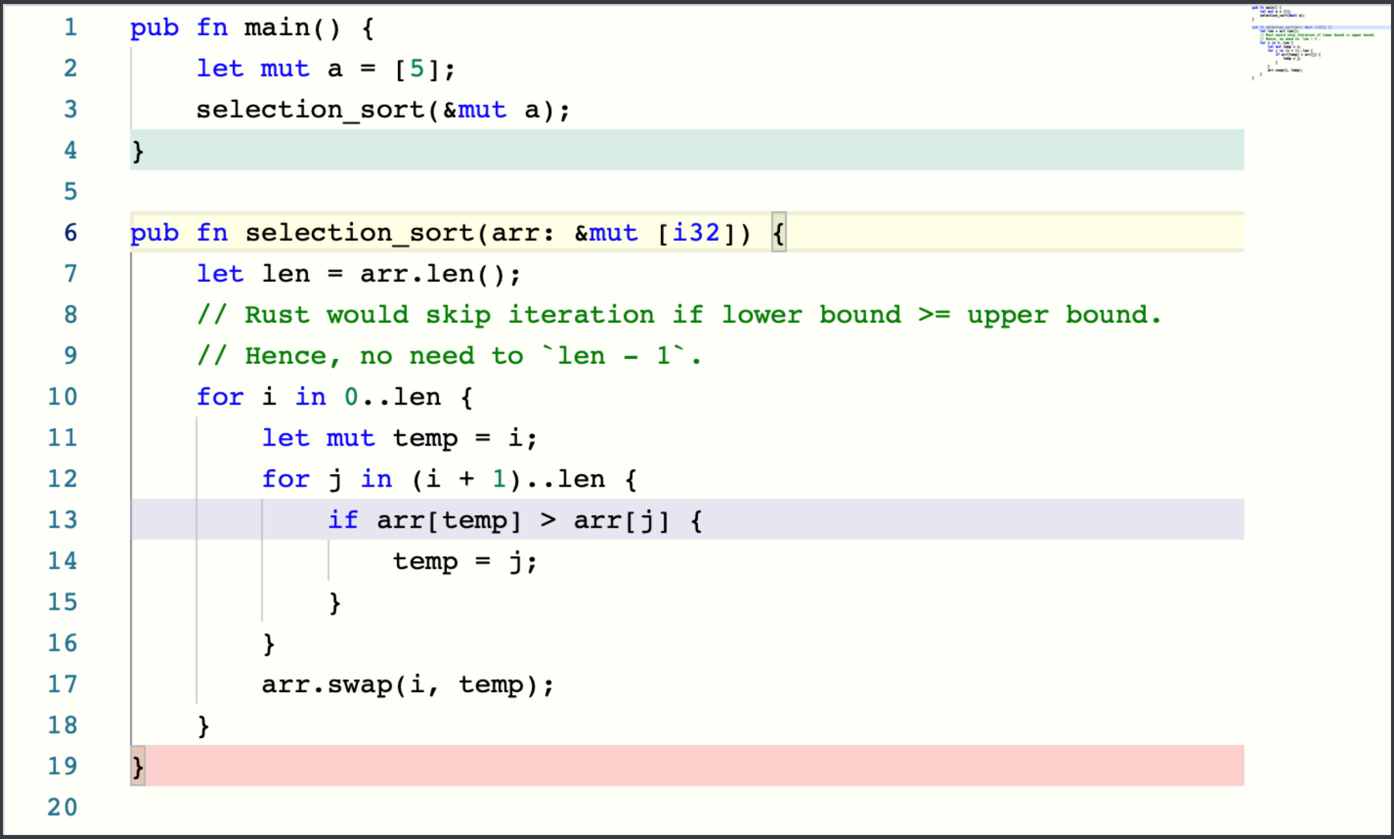
Question 3

```
example::selection_sort:
        sub     rsp, 200
        mov     qword ptr [rsp + 64], rdi
        mov     qword ptr [rsp + 72], rsi
        mov     qword ptr [rsp + 96], 0
        mov     qword ptr [rsp + 104], rsi
        mov     rdi, qword ptr [rsp + 96]
        mov     rsi, qword ptr [rsp + 104]
        call    qword ptr [rip + <I as
core::iter::traits::collect::IntoIterator>::into_iter@GOTPCREL]
        mov     qword ptr [rsp + 80], rax
        mov     qword ptr [rsp + 88], rdx
        mov     rax, qword ptr [rsp + 88]
        mov     rcx, qword ptr [rsp + 80]
        mov     qword ptr [rsp + 112], rcx
        mov     qword ptr [rsp + 120], rax
```

Question 4

With flag `-O`

```
example::selection_sort:
        push    rbx
        test    rsi, rsi
        je      .LBB1_8
        mov     r8b, 1
        xor     r11d, r11d
```



```rust
pub fn main() {
    let mut a = [5];
    selection_sort(&mut a);
}

pub fn selection_sort(arr: &mut [i32]) {
    let len = arr.len();
    // Rust would skip iteration if lower bound >= upper bound.
    // Hence, no need to `len - 1`.
    for i in 0..len {
        let mut temp = i;
        for j in (i + 1)..len {
            if arr[temp] > arr[j] {
                temp = j;
            }
        }
        arr.swap(i, temp);
    }
}
```

Flag `-0` will make the compiler skip the method. As picture shown above, the compiler doesn't call the function `selection_sort()` .

Question 5

```rust
use criterion::{black_box, criterion_group, criterion_main, Criterion};
use rand::Rng;

fn selection_sort(arr: &mut [i64]) {
    let len = arr.len();
    for i in 0..len {
        let mut temp = i;
        for j in (i + 1)..len {
            if arr[temp] > arr[j] {
                temp = j;
            }
        }
        arr.swap(i, temp);
    }
}
```

```
fn criterion_benchmark(c: &mut Criterion) {
    let mut rng = rand::thread_rng();
    let mut l: Vec<i64> = (0..10000).map(|_| {rng.gen_range(1,
10000)}).collect();
    c.bench_function("selection_sort", |b| b.iter(||
selection_sort(black_box(&mut l))));
}

criterion_group!(benches, criterion_benchmark);
criterion_main!(benches);
```

Question 6

```
Gnuplot not found, using plotters backend
selection_sort          time:   [33.844 ms 34.032 ms 34.241 ms]

Found 1 outliers among 100 measurements (1.00%)
  1 (1.00%) high mild
```

Part 2: Code Optimization

Question 7

```rust
// min_by_key to find the minimum's index and the method swap defined in
slices
fn selection_sort_optimize(array: &mut [i64]) {
    for i in 0..array.len() {
        if let Some((j, _)) = array.iter()
            .enumerate()
            .skip(i)
            .min_by_key(|x| x.1) {
            array.swap(i, j);
        }
    }
}
```

Question 8

```rust
fn criterion_benchmark(c: &mut Criterion) {
    let mut rng = rand::thread_rng();
    let mut l: Vec<i64> = (0..10000).map(|_| {rng.gen_range(1,
10000)}).collect();
    c.bench_function("selection_sort_optimize", |b| b.iter(||
selection_sort_optimize(black_box(&mut l))));
}
```

```
selection_sort_optimize time:   [177.50 ms 180.79 ms 184.70 ms]

Found 11 outliers among 100 measurements (11.00%)
  10 (10.00%) high mild
  1 (1.00%) high severe
```

Question 9

For the original one, which contains double `for` loop, the time it needs is around 35 ms. However, for the optimized one, which contains `.map()`, it takes 180 ms.

> The explanation: This is not true as-is. Iterators are not magic. A given piece of code may be slightly faster when written using an iterator (when the iterator allows for elision of bounds checks, as someone already mentioned here), or slightly slower (when the compiler has trouble optimizing away all the generic code they come with). In general, looping and iterators have approximately the same performance characteristics *on average.* - H2CO3 from users.rust-lang.org

Question 10

Simply put, your style of code doesn't affect the performance. In other word, we can say that no additional runtime costs are introduced when using abstraction. It doesn't matter if you use loops or closures, they all compile down to the same assembly.

Part 3: Managing Databases

Question 11

Test function for `test_user()` :

Add two users: Mike and Tim

```
#[test]
pub fn test_user() {
    use super::*;
    use sqlite::State;

    let init_db = UserBase {
        fname: String::from("./data/users.db")
    };

    // clear the database
    init_db.clear_database();

    // pub fn add_user(&self, u_name: &str, p_word: &str)
    let u_name = "Mike";
    let p_word = "123456";

    // Add User "Mike" to database
    init_db.add_user(u_name, p_word);

    // establish a connection to prepare for the next step
    let connection = sqlite::open(&init_db.fname).unwrap();
    // Check whether "Mike" is in database or not, and whether the info
is correct
    let mut st = connection.prepare("select * from users where
u_name=?").unwrap();
    st.bind(1, "Mike").unwrap();

    while let State::Row = st.next().unwrap() {
        // user_name(input) = user_name(db)?
        assert_eq!(String::from(u_name), st.read::<String>(0).unwrap());
        //user_password(input) = user_password(db)?
        let password = verify(p_word, &st.read::<String>(1).unwrap());
        assert_eq!(password.unwrap(), true);
    }

    let u_name_2 = "Tim";
    let p_word_2 = "123456";

    // Add User "Tim" to database
    init_db.add_user(u_name_2, p_word_2);
}
```

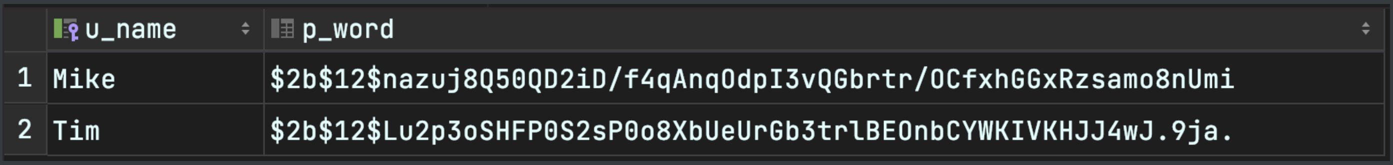
Test function for `test_trans()` : (test for payment)

Make one payment: Mike give Tim $100

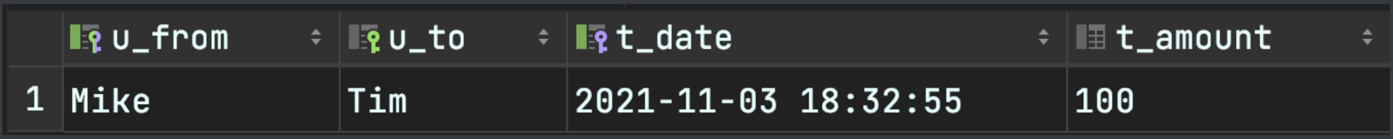
```
#[test]
pub fn test_trans() {
    use super::*;
    use sqlite::State;
    let init_db = UserBase {
        fname: String::from("./data/users.db")
    };

    // pub fn pay(&self, u_from: &str, u_to: &str, amount: i64)
    let u_from = "Mike";
    let u_to = "Tim";
    init_db.pay(u_from, u_to, 100); // Mike gives Tim $100.

    // establish a connection to prepare for the next step
    let connection = sqlite::open(&init_db.fname).unwrap();
    // Check whether the transaction is recorded correctly
    let mut st_2 = connection.prepare("select * from transactions where u_from=? and u_to=?").unwrap();
    st_2.bind(1, "Mike").unwrap();
    st_2.bind(2, "Tim").unwrap();

    while let State::Row = st_2.next().unwrap() {
        assert_eq!(String::from(u_from), st_2.read::<String>(0).unwrap());
        assert_eq!(String::from(u_to), st_2.read::<String>(1).unwrap());
        assert_eq!(100.to_string(), st_2.read::<String>(3).unwrap());
    }
}
```

```
> cargo test test_user
running 1 test
test tests::test_user ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 1 filtered
out; finished in 5.67s

> cargo test test_trans
running 1 test
test tests::test_trans ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 1 filtered
out; finished in 0.01s
```

Cross-validation with database:



	u_name	p_word
1	Mike	$2b$12$nazuj8Q50QD2iD/f4qAnqOdpI3vQGbrtr/OCfxhGGxRzsamo8nUmi
2	Tim	$2b$12$Lu2p3oSHFP0S2sP0o8XbUeUrGb3trlBEOnbCYWKIVKHJJ4wJ.9ja.



	u_from	u_to	t_date	t_amount
1	Mike	Tim	2021-11-03 18:32:55	100