# گزارش تمرین سوم

تینا صداقت ۹۳۳۱۰۴۴

الگوریتم merge sort ۳ گام دارد:

۱. Divide step: که اگر آرایه ۰ یا ۱ خانه دارد بازگشت می کند چون مرتب شده است. در غیر این صورت آرایه ی $A[p .. r]$ را به دو آرایه ی $A[p .. q]$ و $A[q + 1 .. r]$ تقسیم می کند.
۲. Conquer step: به صورت بازگشتی دو آرایه ی $A[p .. q]$ و $A[q + 1 .. r]$ را مرتب می کند.
۳. Combine step: دو آرایه ی $A[p .. q]$ و $A[q + 1 .. r]$ را ترکیب می کند.

می توانیم قسمت conquer step (قسمتی که آرایه به طور بازگشتی در دو زیرآرایه ی راست و چپ مرتب می شود) را موازی سازی کنیم.

برای موازی سازی الگوریتم merge sort دو روش زیر انجام شده است:

**۱. مبتنی بر section:**

قسمتی از کد که قرار است روی دو آرایه ی راست و چپ ایجاد شده، الگوریتم sort را اجرا کند، موازی می کنیم. به این صورت که کل این block از کد داخل #pragma omp parallel قرار می گیرد و داخل دستور pargam omp sections num_threads(8) دو تا section برای هر آرایه تقسیم شده، تعیین می کنیم.

```
#pragma omp parallel
    {

#pragma omp sections num_threads(8)
        {

#pragma omp section
            {
                mergeSort(a, m);
            }
#pragma omp section
            {
                mergeSort(a + m, n - m);
            }
            merge(a, n, m);
        }
    }
```

با جایگذاری اعداد ۲ و ۴ و ۸ در تابع num_threads() به ازای تعداد thread های متفاوت، و همچنین به ازای سایزn های مختلف برای آرایه، برنامه را اجرا می کنیم. توجه داریم که این نتایج با کامپیوتر های آزمایشگاه به دست آمده است که تعداد core آنها ۴ است:

| تعداد نخ ها | 1MB(250000) | تسریع | 10MB(2500000) | تسریع | 100MB(25000000) | تسریع | 1GB(250000000) | تسریع |
|---|---|---|---|---|---|---|---|---|
| ۱ | ۱۳۰,۵۹ | - | ۱۱۷۰,۱۳ | - | ۱۲۴۳۱,۵۸ | - | ۱۳۰۶۵۰,۱۹ | - |
| ۲ | ۱۱۵,۵۸ | ۱,۱ | ۱۱۶۵,۶۱ | ۱,۰ | ۱۲۳۷۹,۸۴ | ۱,۰ | ۱۳۰۵۹۹,۲۰ | ۱,۰۰۳ |
| ۴ | ۱۵۴,۱۱ | ۰,۸۴ | ۱۱۷۴,۱۷ | ۰,۹۹ | ۱۲۳۰۱,۷۲ | ۱,۰۱ | ۱۳۰۶۶۲,۷۴ | ۰,۹۹ |
| ۸ | ۱۲۰,۱ | ۱,۰۸ | ۱۱۶۹,۱۵ | ۰,۹۹ | ۱۲۳۴۲,۸۸ | ۱,۰ | ۱۳۴۷۸۵,۸۳ | ۰,۹۶ |

همانطور که می بینیم تسریع چندانی بدست نیامده است و معمولا به ازای ۲ تا thread مقدار کمی تسریع گرفته ایم. چون سربار محاسباتی زیاد است و بایدthread ها برای کار یکدیگر صبر کنند.

## ۲. مبتنی بر task:

نحوه پیاده سازی task بسیار شبیه به Section است با این تفاوت که در Section ، thread ها صبر می کنند تا کار همه ی Thread ها تمام شود و سپس از بلاک omp sections بیرون می آیند.(در واقع implicit barrier وجود دارد.)

اما در task، Task ها در صف قرار می گیرند وهر وقت که ممکن باشد سر زمان هایی اجرا می شوند.

```
#pragma omp parallel task num_threads(8)
    {
#pragma omp single
            {
#pragma omp task
        {
            mergeSort(a, m);
        }
#pragma omp task
        {
            mergeSort(a + m, n - m);
        }
        merge(a, n, m);
    }
    }
```

| تعداد نخ ها | 1MB(250000) | تسریع | 10MB(2500000) | تسریع | 100MB(25000000) | تسریع | 1GB(250000000) | تسریع |
|---|---|---|---|---|---|---|---|---|
| ۱ | ۱۱۴,۷۴ | - | ۱۱۶۱,۲۱ | - | ۱۲۳۷۲,۷۱ | - | ۱۲۹۶۸۵,۵۸ | - |
| ۲ | ۱۲۲,۰۲ | ۰,۹۴ | ۱۱۹۳,۵۴ | ۰,۹۷ | ۱۲۳۴۵,۱۰ | ۱,۰۵ | ۱۲۹۴۳۸,۷۴ | ۱,۰۰۱ |
| ۴ | ۱۲۵,۱۵ | ۰,۹۲ | ۱۲۰۹,۲۶ | ۰,۹۶ | ۱۲۳۸۳,۵۶ | ۰,۹۹ | ۱۲۹۶۹۹,۴۴ | ۰,۹۹ |
| ۸ | ۱۱۴,۸۹ | ۱,۰ | ۱۱۷۰,۱۲ | ۰,۹۹ | ۱۲۳۰۲,۷۴ | ۱,۰۵ | ۱۳۳۳۹۸,۵۹ | ۰,۹۷ |

همانطور که می بینیم تسریع چندانی بدست نیامده است چون thread ها باید برای همدیگر صبر کنند و سربار هم زیاد شده است.