Project 3

Checkoff Item 1：

> Make note of one bottleneck.

在虚拟机上安装perf

安装好开始运行时出现如下报错

```
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2$ perf record ./isort 10000 10
Error:
Access to performance monitoring and observability operations is limited.
Consider adjusting /proc/sys/kernel/perf_event_paranoid setting to open
access to performance monitoring and observability operations for processes
without CAP_PERFMON, CAP_SYS_PTRACE or CAP_SYS_ADMIN Linux capability.
More information can be found at 'Perf events and tool security' document:
https://www.kernel.org/doc/html/latest/admin-guide/perf-security.html
perf_event_paranoid setting is 4:
  -1: Allow use of (almost) all events by all users
      Ignore mlock limit after perf_event_mlock_kb without CAP_IPC_LOCK
>= 0: Disallow raw and ftrace function tracepoint access
>= 1: Disallow CPU event access
>= 2: Disallow kernel profiling
To make the adjusted perf_event_paranoid setting permanent preserve it
in /etc/sysctl.conf (e.g. kernel.perf_event_paranoid = <setting>)
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2$
```

这是因为出于安全考虑，内核在默认情况下禁止非特权用户监测系统性能。解决的办法有以下几种：

- 修改 kernel.perf_event_paranoid 内核参数
- 给 perf 加入 CAP_PERFMON 权限位
- 使用 root 用户进行性能监测

这里修改内核参数

修改后运行perf record出现如下warning

```
WARNING: Kernel address maps (/proc/{kallsyms,modules}) are restricted,
check /proc/sys/kernel/kptr_restrict and /proc/sys/kernel/perf_event_paranoid.

Samples in kernel functions may not be resolved if a suitable vmlinux
file is not found in the buildid cache or in the vmlinux path.

Samples in kernel modules won't be resolved at all.

If some relocation was applied (e.g. kexec) symbols may be misresolved
even with a suitable vmlinux or kallsyms file.

Couldn't record kernel reference relocation symbol
Symbol resolution may be skewed if relocation was used (e.g. kexec).
Check /proc/kallsyms permission or run as root.
```

运行如下命令

```
sudo sh -c " echo 0 > /proc/sys/kernel/kptr_restrict"
```

运行成功，结果如下

```
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/recitation$ make isort DEBUG=1
clang -Wall -O0 -g -c isort.c
clang -Wall -O0 -g -c qsort.c
clang -o isort isort.o qsort.o -lrt
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/recitation$ ls
isort  isort.c  isort.o  Makefile  qsort.c  qsort.o  sum.c  verifier.py
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/recitation$ perf record ./isort 100
00 10
Sorting 10000 values...
Done!
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.086 MB perf.data (1958 samples) ]
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/recitation$
```

运行perf report

```
Samples: 1K of event 'cpu-clock:pppH', Event count (approx.): 489500000
Overhead  Command  Shared Object      Symbol
  99.64%  isort    isort              [.] isort
   0.10%  isort    libc-2.31.so       [.] rand_r
   0.05%  isort    [kernel.kallsyms]  [k] __d_lookup_rcu
   0.05%  isort    [kernel.kallsyms]  [k] rcu_core
   0.05%  isort    isort              [.] main
   0.05%  isort    libc-2.31.so       [.] __vfprintf_internal
   0.05%  isort    libc-2.31.so       [.] _dl_addr
```

会有Cannot load tips.txt file, please install perf!的报错

解决方法在该路径下保存正确的tips.txt即可

```
wangwenqing@ubuntu:/usr/share/doc/perf-tip$ sudo touch tips.txt
wangwenqing@ubuntu:/usr/share/doc/perf-tip$ ls
tips.txt
wangwenqing@ubuntu:/usr/share/doc/perf-tip$ vim tips.txt
wangwenqing@ubuntu:/usr/share/doc/perf-tip$ sudo vim tips.txt
```

可以看到瓶颈在isort这里

更细化的看具体函数，命令如下

```
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/recitation$ perf record -e cpu-clock -g ./isort 10000 10
Sorting 10000 values...
Done!
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.209 MB perf.data (2584 samples) ]
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/recitation$ perf report -g
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/recitation$
```

```
Samples: 2K of event 'cpu-clock', Event count (approx.): 646000000
  Children      Self  Command  Shared Object       Symbol
+   99.92%     0.00%  isort    libc-2.31.so        [.] __libc_start_main
+   99.88%     0.08%  isort    isort               [.] main
+   99.81%    99.57%  isort    isort               [.] isort
     0.15%     0.00%  isort    [kernel.kallsyms]   [k] irq_exit_rcu
     0.15%     0.00%  isort    [kernel.kallsyms]   [k] do_softirq_own_stack
     0.15%     0.00%  isort    [kernel.kallsyms]   [k] asm_call_sysvec_on_stack
     0.15%     0.08%  isort    [kernel.kallsyms]   [k] __softirqentry_text_start
     0.12%     0.00%  isort    [kernel.kallsyms]   [k] asm_common_interrupt
     0.12%     0.00%  isort    [kernel.kallsyms]   [k] common_interrupt
     0.12%     0.00%  isort    [kernel.kallsyms]   [k] asm_sysvec_apic_timer_interrupt
     0.12%     0.00%  isort    [kernel.kallsyms]   [k] sysvec_apic_timer_interrupt
     0.08%     0.08%  isort    [kernel.kallsyms]   [k] __lock_text_start
     0.08%     0.04%  isort    [kernel.kallsyms]   [k] schedule
     0.08%     0.00%  isort    [kernel.kallsyms]   [k] asm_exc_page_fault
     0.08%     0.00%  isort    [kernel.kallsyms]   [k] irqentry_exit
     0.08%     0.00%  isort    [kernel.kallsyms]   [k] exc_page_fault
     0.08%     0.00%  isort    [kernel.kallsyms]   [k] irqentry_exit_to_user_mode
     0.04%     0.00%  isort    [kernel.kallsyms]   [k] blk_done_softirq
     0.04%     0.04%  isort    [kernel.kallsyms]   [k] finish_task_switch
     0.04%     0.00%  isort    [kernel.kallsyms]   [k] scsi_softirq_done
     0.04%     0.00%  isort    [kernel.kallsyms]   [k] scsi_finish_command
     0.04%     0.00%  isort    [kernel.kallsyms]   [k] scsi_io_completion
     0.04%     0.00%  isort    [kernel.kallsyms]   [k] scsi_end_request
     0.04%     0.04%  isort    [kernel.kallsyms]   [k] do_fault
     0.04%     0.04%  isort    [kernel.kallsyms]   [k] up_read
     0.04%     0.00%  isort    [kernel.kallsyms]   [k] blk_update_request
     0.04%     0.00%  isort    [kernel.kallsyms]   [k] run_timer_softirq
     0.04%     0.00%  isort    libc-2.31.so        [.] _exit
Tip: Show current config key-value pairs: perf config --list
```

可以看到在整个isort运行中isort函数占了大头，为主要优化对象

运行lscpu

```
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/recitation$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   43 bits physical, 48 bits virtual
CPU(s):                          1
On-line CPU(s) list:             0
Thread(s) per core:              1
Core(s) per socket:              1
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           142
Model name:                      Intel(R) Core(TM) i5-8279U CPU @ 2.40GHz
Stepping:                        10
CPU MHz:                         2399.074
BogoMIPS:                        4798.14
Hypervisor vendor:               VMware
Virtualization type:             full
L1d cache:                       32 KiB
L1i cache:                       32 KiB
L2 cache:                        256 KiB
L3 cache:                        6 MiB
NUMA node0 CPU(s):               0
Vulnerability Itlb multihit:     KVM: Mitigation: VMX unsupported
Vulnerability L1tf:              Mitigation; PTE Inversion
Vulnerability Mds:               Mitigation; Clear CPU buffers; SMT Host state u
                                 nknown
Vulnerability Meltdown:          Mitigation; PTI
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled v
```

```
Vulnerability Mds:                Mitigation; Clear CPU buffers; SMT Host state u
                                  nknown
Vulnerability Meltdown:           Mitigation; PTI
Vulnerability Spec store bypass:  Mitigation; Speculative Store Bypass disabled v
                                  ia prctl and seccomp
Vulnerability Spectre v1:         Mitigation; usercopy/swapgs barriers and __user
                                   pointer sanitization
Vulnerability Spectre v2:         Mitigation; Full generic retpoline, IBPB condit
                                  ional, IBRS_FW, STIBP disabled, RSB filling
Vulnerability Srbds:              Unknown: Dependent on hypervisor status
Vulnerability Tsx async abort:    Not affected
Flags:                            fpu vme de pse tsc msr pae mce cx8 apic sep mtr
                                  r pge mca cmov pat pse36 clflush mmx fxsr sse s
                                  se2 ss syscall nx pdpe1gb rdtscp lm constant_ts
                                  c arch_perfmon nopl xtopology tsc_reliable nons
                                  top_tsc cpuid pni pclmulqdq ssse3 fma cx16 pcid
                                   sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline
                                  _timer aes xsave avx f16c rdrand hypervisor lah
                                  f_lm abm 3dnowprefetch cpuid_fault invpcid_sing
                                  le pti ssbd ibrs ibpb stibp fsgsbase tsc_adjust
                                   bmi1 avx2 smep bmi2 invpcid rdseed adx smap cl
                                  flushopt xsaveopt xsavec xsaves arat md_clear f
                                  lush_l1d arch_capabilities
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/recitation$
```

另外我还运行了perf top以及perf stat来查看相关结果

运行perf top

```
Samples: 3K of event 'cpu-clock:pppH', 4000 Hz, Event count (approx.): 350037039 lost: 0/0 drop: 0/0
Overhead  Shared Object                     Symbol
  14.23%  [kernel]                          [k] clear_page_orig
   6.04%  [kernel]                          [k] finish_task_switch
   4.31%  [kernel]                          [k] cpuidle_enter_state
   3.91%  [kernel]                          [k] __lock_text_start
   2.33%  perf                              [.] rb_next
   1.67%  [kernel]                          [k] vmware_sched_clock
   1.50%  libc-2.31.so                      [.] _int_malloc
   1.20%  [kernel]                          [k] kallsyms_expand_symbol.constprop.0
   1.14%  [kernel]                          [k] collect_percpu_times
   1.13%  perf                              [.] evsel__parse_sample
   1.03%  perf                              [.] kallsyms__parse
   0.96%  perf                              [.] dso__find_symbol
   0.94%  [kernel]                          [k] copy_user_generic_unrolled
   0.90%  perf                              [.] map__process_kallsym_symbol
   0.90%  [vmwgfx]                          [k] vmw_cmdbuf_header_submit
   0.80%  perf                              [.] __symbols__insert
   0.73%  [mptbase]                         [k] mpt_put_msg_frame
   0.73%  libc-2.31.so                      [.] __libc_calloc
   0.67%  [kernel]                          [k] __softirqentry_text_start
   0.67%  [kernel]                          [k] format_decode
   0.66%  libc-2.31.so                      [.] sysmalloc
   0.61%  libc-2.31.so                      [.] __strcmp_avx2
   0.60%  [kernel]                          [k] do_user_addr_fault
   0.59%  libc-2.31.so                      [.] malloc
   0.57%  libc-2.31.so                      [.] _int_free
   0.56%  [kernel]                          [k] menu_reflect
   0.51%  perf                              [.] deliver_event
   0.50%  [kernel]                          [k] __run_timers.part.0
For a higher level overview, try: perf top --sort comm,dso
```

可以看到clear_page_orig以及finish_task_switch占用cpu资源的比例较大

运行perf stat

```
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/recitation$ sudo perf stat ./isort
[sudo] password for wangwenqing:
Error: wrong number of arguments.

 Performance counter stats for './isort':

              0.71 msec task-clock                #    0.299 CPUs utilized
                 3      context-switches          #    0.004 M/sec
                 0      cpu-migrations            #    0.000 K/sec
                66      page-faults               #    0.092 M/sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

       0.002388140 seconds time elapsed

       0.000000000 seconds user
       0.002229000 seconds sys
```

Checkoff Item 2:

> Run sum under cachegrind to identify cache performance. It may take a little while. In the output, look at the D1 and LLd misses. D1 represents the lowest-level cache (L1), and LL represents the last (highest) level data cache (on most machines, L3). Do these numbers correspond with what you would expect? Try playing around with the values N and U in sum.c. How can you bring down the number of cache misses?

在cache grind 下运行sum

```
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/recitation$ valgrind --tool=cachegrind --branch-sim=yes ./s
um
==59260== Cachegrind, a cache and branch-prediction profiler
==59260== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==59260== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==59260== Command: ./sum
==59260==
--59260-- warning: L3 cache found, using its data for the LL simulation.
Allocated array of size 10000000
Summing 100000000 random values...
Done. Value = 938895920
==59260==
==59260== I   refs:      3,540,247,565
==59260== I1  misses:            1,219
==59260== LLi misses:            1,207
==59260== I1  miss rate:          0.00%
==59260== LLi miss rate:          0.00%
==59260==
==59260== D   refs:        610,075,004  (400,058,062 rd   + 210,016,942 wr)
==59260== D1  misses:      100,548,142  ( 99,922,251 rd   +     625,891 wr)
==59260== LLd misses:       84,954,029  ( 84,328,207 rd   +     625,822 wr)
==59260== D1  miss rate:          16.5% (       25.0%     +         0.3%  )
==59260== LLd miss rate:          13.9% (       21.1%     +         0.3%  )
==59260==
==59260== LL refs:         100,549,361  ( 99,923,470 rd   +     625,891 wr)
==59260== LL misses:        84,955,236  ( 84,329,414 rd   +     625,822 wr)
==59260== LL miss rate:            2.0% (        2.1%     +         0.3%  )
==59260==
==59260== Branches:        210,048,916  (110,048,412 cond + 100,000,504 ind)
==59260== Mispredicts:           5,753  (      5,539 cond +         214 ind)
==59260== Mispred rate:            0.0% (        0.0%     +         0.0%   )
```

D1misses(代表了L1 cache的miss) 100548142，比例为16.5%

LLD misses(大多数情况代表L3 cache的miss) 84954029，比例为13.9%

运行lscpu

可以看到l1d cache 32KiB L2cache 256Kib L3 cache 6MiB

```
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/recitation$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   43 bits physical, 48 bits virtual
CPU(s):                          1
On-line CPU(s) list:             0
Thread(s) per core:              1
Core(s) per socket:              1
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           142
Model name:                      Intel(R) Core(TM) i5-8279U CPU @ 2.40GHz
Stepping:                        10
CPU MHz:                         2399.074
BogoMIPS:                        4798.14
Hypervisor vendor:               VMware
Virtualization type:             full
L1d cache:                       32 KiB
L1i cache:                       32 KiB
L2 cache:                        256 KiB
L3 cache:                        6 MiB
NUMA node0 CPU(s):               0
Vulnerability Itlb multihit:     KVM: Mitigation: VMX unsupported
```

```
Vulnerability L1tf:              Mitigation; PTE Inversion
Vulnerability Mds:               Mitigation; Clear CPU buffers; SMT Host state unknown
Vulnerability Meltdown:          Mitigation; PTI
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl and seccomp
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2:        Mitigation; Full generic retpoline, IBPB conditional, IBRS_FW, STIBP di
                                 sabled, RSB filling
Vulnerability Srbds:             Unknown: Dependent on hypervisor status
Vulnerability Tsx async abort:   Not affected
Flags:                           fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
                                  clflush mmx fxsr sse sse2 ss syscall nx pdpe1gb rdtscp lm constant_tsc
                                  arch_perfmon nopl xtopology tsc_reliable nonstop_tsc cpuid pni pclmulq
                                 dq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_t
                                 imer aes xsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch cpu
                                 id_fault invpcid_single pti ssbd ibrs ibpb stibp fsgsbase tsc_adjust bm
                                 i1 avx2 smep bmi2 invpcid rdseed adx smap clflushopt xsaveopt xsavec xs
                                 aves arat md_clear flush_l1d arch_capabilities
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/recitation$
```

要降低miss rate，需要减小随机数范围U的数值

由于L3 Cache大小是6M，令U=1,000,000时数组大小约为4M

运行结果如下

```
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/recitation$ make sum
clang -Wall -O1 -DNDEBUG  -c sum.c
clang -o sum sum.o -lrt
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/recitation$ valgrind --tool=cachegrind --branch-sim=yes ./s
um
==64758== Cachegrind, a cache and branch-prediction profiler
==64758== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==64758== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==64758== Command: ./sum
==64758==
--64758-- warning: L3 cache found, using its data for the LL simulation.
Allocated array of size 1000000
Summing 100000000 random values...
Done. Value = 369800944
==64758==
==64758== I   refs:      3,504,247,558
==64758== I1  misses:            1,220
==64758== LLi misses:            1,195
==64758== I1  miss rate:          0.00%
==64758== LLi miss rate:          0.00%
==64758==
==64758== D   refs:        601,075,005  (400,058,062 rd   + 201,016,943 wr)
==64758== D1  misses:       99,249,231  ( 99,185,840 rd   +      63,391 wr)
==64758== LLd misses:           65,565  (      2,309 rd   +      63,256 wr)
==64758== D1  miss rate:          16.5% (       24.8%     +         0.0%  )
==64758== LLd miss rate:           0.0% (        0.0%     +         0.0%  )
==64758==
==64758== LL refs:          99,250,451  ( 99,187,060 rd   +      63,391 wr)
==64758== LL misses:            66,760  (      3,504 rd   +      63,256 wr)
==64758== LL miss rate:            0.0% (        0.0%     +         0.0%  )
==64758==
```

可以看到三级缓存的miss rate变为0

反之比如尝试将U的值扩大20倍,看到miss rate没有很大的变化

```
==59613== Cachegrind, a cache and branch-prediction profiler
==59613== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==59613== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==59613== Command: ./sum
==59613==
--59613-- warning: L3 cache found, using its data for the LL simulation.
Allocated array of size 200000000
Summing 100000000 random values...
Done. Value = -727868112
==59613==
==59613== I   refs:      4,300,247,662
==59613== I1  misses:            1,226
==59613== LLi misses:            1,214
==59613== I1  miss rate:          0.00%
==59613== LLi miss rate:          0.00%
==59613==
==59613== D   refs:        800,075,040  (400,058,085 rd   + 400,016,955 wr)
==59613== D1  misses:      112,500,101  ( 99,999,210 rd   +  12,500,891 wr)
==59613== LLd misses:      111,739,014  ( 99,238,192 rd   +  12,500,822 wr)
==59613== D1  miss rate:          14.1% (       25.0%     +         3.1%  )
==59613== LLd miss rate:          14.0% (       24.8%     +         3.1%  )
==59613==
==59613== LL refs:         112,501,327  (100,000,436 rd   +  12,500,891 wr)
==59613== LL misses:       111,740,228  ( 99,239,406 rd   +  12,500,822 wr)
==59613== LL miss rate:           2.2% (        2.1%     +         3.1%  )
==59613==
==59613== Branches:        400,048,934  (300,048,429 cond + 100,000,505 ind)
==59613== Mispredicts:           5,766  (      5,551 cond +         215 ind)
==59613== Mispred rate:            0.0% (        0.0%     +         0.0%   )
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/recitation$
```

write up 1

> Compare the Cachegrind output on the DEBUG=1 code versus DEBUG=0 compiler optimized code. Explain the advantages and disadvantages of using instruction count as a substitute for time when you compare the performance of different versions of this program.

比较 Cachegrind 结果的不同on the DEBUG=1 code versus DEBUG=0 compiler optimized code

DEBUG=0

```
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/homework_new$ make
clang main.c tests.c util.c isort.c sort_a.c sort_c.c sort_i.c sort_p.c sort_m.c sort_f.c -O3 -DNDEBUG -
g -Wall -std=gnu99 -gdwarf-3 -always-inline -lrt -lm  -o sort
clang: warning: argument unused during compilation: '-always-inline' [-Wunused-command-line-argument]
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/homework_new$ valgrind --tool=cachegrind --branch-sim=yes .
/sort 100000 100
==65111== Cachegrind, a cache and branch-prediction profiler
==65111== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==65111== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==65111== Command: ./sort 100000 100
==65111==
--65111-- warning: L3 cache found, using its data for the LL simulation.

Running test #0...
Generating random array of 100000 elements
Arrays are sorted: yes
 --> test_correctness at line 217: PASS
sort_a          : Elapsed execution time: 0.278941 sec
sort_a repeated : Elapsed execution time: 0.278582 sec
Generating inverted array of 100000 elements
Arrays are sorted: yes
 --> test_correctness at line 217: PASS
sort_a          : Elapsed execution time: 0.537967 sec
sort_a repeated : Elapsed execution time: 0.539905 sec

Running test #1...
 --> test_zero_element at line 245: PASS
```

```
Running test #2...
 --> test_one_element at line 266: PASS
Done testing.
==65111==
==65111== I   refs:      24,679,756,350
==65111== I1  misses:             1,613
==65111== LLi misses:             1,517
==65111== I1  miss rate:           0.00%
==65111== LLi miss rate:           0.00%
==65111==
==65111== D   refs:       9,259,541,340  (5,643,703,226 rd   + 3,615,838,114 wr)
==65111== D1  misses:        64,072,064  (   33,465,920 rd   +    30,606,144 wr)
==65111== LLd misses:            28,338  (        2,468 rd   +        25,870 wr)
==65111== D1  miss rate:            0.7% (          0.6%     +           0.8%   )
==65111== LLd miss rate:            0.0% (          0.0%     +           0.0%   )
==65111==
==65111== LL refs:           64,073,677  (   33,467,533 rd   +    30,606,144 wr)
==65111== LL misses:             29,855  (        3,985 rd   +        25,870 wr)
==65111== LL miss rate:             0.0% (          0.0%     +           0.0%   )
==65111==
==65111== Branches:       4,071,679,156  (3,901,676,299 cond +   170,002,857 ind)
==65111== Mispredicts:      308,988,098  (  308,987,742 cond +           356 ind)
==65111== Mispred rate:             7.6% (          7.9%     +           0.0%   )
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/homework_new$
```

DEBUG=1

```
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/homework_new$ make DEBUG=1
clang main.c tests.c util.c isort.c sort_a.c sort_c.c sort_i.c sort_p.c sort_m.c sort_f.c -DDEBUG -O0 -g
 -Wall -std=gnu99 -gdwarf-3 -always-inline -lrt -lm  -o sort
clang: warning: argument unused during compilation: '-always-inline' [-Wunused-command-line-argument]
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/homework_new$ valgrind --tool=cachegrind --branch-sim=yes .
/sort 100000 100
==65157== Cachegrind, a cache and branch-prediction profiler
==65157== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==65157== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==65157== Command: ./sort 100000 100
==65157==
--65157-- warning: L3 cache found, using its data for the LL simulation.

Running test #0...
Generating random array of 100000 elements
Arrays are sorted: yes
 --> test_correctness at line 217: PASS
sort_a          : Elapsed execution time: 0.462529 sec
sort_a repeated : Elapsed execution time: 0.463225 sec
Generating inverted array of 100000 elements
Arrays are sorted: yes
 --> test_correctness at line 217: PASS
sort_a          : Elapsed execution time: 0.914391 sec
sort_a repeated : Elapsed execution time: 0.915680 sec

Running test #1...
 --> test_zero_element at line 245: PASS
```

```
Running test #2...
 --> test_one_element at line 266: PASS
Done testing.
==65157==
==65157== I   refs:      46,453,739,384
==65157== I1  misses:             1,586
==65157== LLi misses:             1,507
==65157== I1  miss rate:           0.00%
==65157== LLi miss rate:           0.00%
==65157==
==65157== D   refs:      30,085,113,009  (22,682,088,199 rd   + 7,403,024,810 wr)
==65157== D1  misses:        64,042,886  (    33,500,613 rd   +    30,542,273 wr)
==65157== LLd misses:            28,326  (         2,457 rd   +        25,869 wr)
==65157== D1  miss rate:            0.2% (           0.1%     +           0.4%  )
==65157== LLd miss rate:            0.0% (           0.0%     +           0.0%  )
==65157==
==65157== LL refs:           64,044,472  (    33,502,199 rd   +    30,542,273 wr)
==65157== LL misses:             29,833  (         3,964 rd   +        25,869 wr)
==65157== LL miss rate:             0.0% (           0.0%     +           0.0%  )
==65157==
==65157== Branches:       4,670,669,406  ( 4,500,666,747 cond +   170,002,659 ind)
==65157== Mispredicts:      395,524,900  (   395,524,554 cond +           346 ind)
==65157== Mispred rate:             8.5% (           8.8%     +           0.0%   )
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/homework_new$
```

通过比较可以看到：

首先指令获取的缓存访问，可以看到DEBUG=1时，执行的指令数减少给出了获取的次数、I1 未命中的比例不变和 LL 指令数（LLi) 未命中的比例降低。

对于数据的缓存，DEBUG=1时总数减少，读取和写入都有减少，而D1未命中比例和LLD未命中比例不变。

DEBUG=1时，分支数减少，而错误预测的比例不变。

衡量性能时用指令数代替时间的利与弊：

利：得到指令数很简单

由于指令数取决于体系结构，而不是确切的实现，因此我们可以在不知道实现的所有细节的情况下测量指令数。

弊端：指令数在一些情况无法准确衡量计算性能。更改指令集以降低指令计数可能会导致一个时钟周期时间变慢或 CPI 更高的系统，从而抵消了指令计数的改进。

write up 2

> Explain which functions you chose to inline and report the performance differences you observed between the inlined and uninlined sorting routines.

内联函数选择，内联函数语句应不超过10行，只适合函数体内代码简单的函数数使用，不能包含复杂的结构控制语句例如while、switch。内联函数本身不能是直接递归函数。

我选择内联copy_i 函数。*mem_alloc*和*mem_free*

内联是以代码膨胀（复制）为代价，仅仅省去了函数调用的开销，从而提高函数的执行效率。如果执行函数体内代码的时间，相比于函数调用的开销较大，那么效率的收获会很少。另一方面，**每一处内联函数的调用都要复制代码，将使程序的总代码量增大，消耗更多的内存空间**。

在main.c将sort_i的注释符删除

```
struct testFunc_t testFunc[] = {
    {&sort_a, "sort_a\t\t"},
    {&sort_a, "sort_a repeated\t"},
    {&sort_i, "sort_i\t\t"},
```

性能测试如下

内联前

```
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/homework_new$ make DEBUG=0
clang main.c tests.c util.c isort.c sort_a.c sort_c.c sort_i.c sort_p.c sort_m.c sort_f.c -O3 -DNDEBUG -
g -Wall -std=gnu99 -gdwarf-3 -always-inline -lrt -lm  -o sort
clang: warning: argument unused during compilation: '-always-inline' [-Wunused-command-line-argument]
```

```
Running test #2...
 --> test_one_element at line 266: PASS
Done testing.
==65111==
==65111== I   refs:      24,679,756,350
==65111== I1  misses:             1,613
==65111== LLi misses:             1,517
==65111== I1  miss rate:           0.00%
==65111== LLi miss rate:           0.00%
==65111==
==65111== D   refs:       9,259,541,340  (5,643,703,226 rd   + 3,615,838,114 wr)
==65111== D1  misses:        64,072,064  (   33,465,920 rd   +    30,606,144 wr)
==65111== LLd misses:            28,338  (        2,468 rd   +        25,870 wr)
==65111== D1  miss rate:            0.7% (          0.6%     +           0.8%   )
==65111== LLd miss rate:            0.0% (          0.0%     +           0.0%   )
==65111==
==65111== LL refs:           64,073,677  (   33,467,533 rd   +    30,606,144 wr)
==65111== LL misses:             29,855  (        3,985 rd   +        25,870 wr)
==65111== LL miss rate:             0.0% (          0.0%     +           0.0%   )
==65111==
==65111== Branches:       4,071,679,156  (3,901,676,299 cond +   170,002,857 ind)
==65111== Mispredicts:      308,988,098  (  308,987,742 cond +           356 ind)
==65111== Mispred rate:             7.6% (          7.9%     +           0.0%   )
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/homework_new$
```

内联后

```
Running test #2...
 --> test_one_element at line 266: PASS
Done testing.
==12558==
==12558== I   refs:      46,454,151,613
==12558== I1  misses:             1,536
==12558== LLi misses:             1,466
==12558== I1  miss rate:           0.00%
==12558== LLi miss rate:           0.00%
==12558==
==12558== D   refs:      30,085,112,932  (22,682,088,463 rd   + 7,403,024,469 wr)
==12558== D1  misses:        63,875,570  (    33,396,835 rd   +    30,478,735 wr)
==12558== LLd misses:            28,328  (         2,465 rd   +        25,863 wr)
==12558== D1  miss rate:            0.2% (           0.1%     +           0.4%  )
==12558== LLd miss rate:            0.0% (           0.0%     +           0.0%  )
==12558==
==12558== LL refs:           63,877,106  (    33,398,371 rd   +    30,478,735 wr)
==12558== LL misses:             29,794  (         3,931 rd   +        25,863 wr)
==12558== LL miss rate:             0.0% (           0.0%     +           0.0%  )
==12558==
==12558== Branches:       4,590,463,296  ( 4,420,460,624 cond +   170,002,672 ind)
==12558== Mispredicts:      388,907,915  (   388,907,559 cond +           356 ind)
==12558== Mispred rate:             8.5% (           8.8%     +           0.0%   )
```

可以看到内联后对性能影响不明显。分析原因可能是优化等级已经设为O3导致inline的优化效果不显著。

write up 3

> Explain the possible performance downsides of inlining recursive functions. How could profiling data gathered using cachegrind help you measure these negative performance effects?

内联递归函数可能出现的性能缺点

因为函数调用是需要开销的（函数调用时的参数压栈、栈帧开辟与销毁、以及寄存器保存与恢复等等操作），因为内联函数是在编译阶段被编译器展开在调用处的，多次调用就需要多段重复的代码放在各个调用处，这样会增加代码量。在空间上，一般来说使用内联函数会导致生成的可执行文件变大。递归调用堆栈的展开并不像循环那么简单, 比如递归层数在编译时可能是未知的, 可能造成代码的无限inline循环。所以如果对递归函数进行强制内联，会得不偿失，运行性能变差。

cachegrind 收集的分析数据如何帮助衡量这些负面性能影响

Cachegrind模拟CPU中的一级缓存和二级缓存，能够精确地指出程序中cache的丢失和命中。

cachegrind收集到的包括高速缓存的命中率，高速缓存包括指令缓存和数据缓存，可以通过看缓存命中率的变化，看出内联函数使用不当带来的负面影响。此外也可以直接从指令数进行分析。

在现代机器上，L1 未命中通常会花费大约 10 个周期，LL 未命中可能会花费多达 200 个周期，而错误预测的分支会花费 10 到 30 个周期。详细的缓存和分支分析可以看到内联后的程序与机器交互中的变化。

write up 4

> Give a reason why using pointers may improve performance. Report on any performance differences you observed in your implementation.

修改为指针,主要修改的部分如下。修改了merge_p和copy_p。

定义两个指针指向left和right并进行依次遍历。

```
  data_t* l_start=left;
  data_t* r_start=right;

  data_t* A0=A+p;
  for (; A0 <= A+r; A0++) {
    if (*l_start <= *r_start) {
      *A0=*l_start;
      l_start++;
    } else {
      *A0=*r_start;
      r_start++;
    }
  }
  mem_free(&left);
  mem_free(&right);
}

static void copy_p(data_t* source, data_t* dest, int n) {
  assert(dest);
  assert(source);

  data_t* s=source;
  data_t* d=dest;
  for (; s< source+n&& d< dest+n ; s++&&d++) {
    *d=*s;
  }
```

```
Arrays are sorted: yes
 --> test_correctness at line 217: PASS
sort_p          : Elapsed execution time: 0.481815 sec

Running test #1...
 --> test_zero_element at line 245: PASS

Running test #2...
 --> test_one_element at line 266: PASS
Done testing.
==66518==
==66518== I   refs:      13,459,922,386
==66518== I1  misses:             1,585
==66518== LLi misses:             1,503
==66518== I1  miss rate:           0.00%
==66518== LLi miss rate:           0.00%
==66518==
==66518== D   refs:       5,068,041,114  (3,040,374,564 rd   + 2,027,666,550 wr)
==66518== D1  misses:        33,919,657  (   17,366,661 rd   +    16,552,996 wr)
==66518== LLd misses:            28,339  (        2,470 rd   +        25,869 wr)
==66518== D1  miss rate:            0.7% (          0.6%     +           0.8%   )
==66518== LLd miss rate:            0.0% (          0.0%     +           0.0%   )
==66518==
==66518== LL refs:           33,921,242  (   17,368,246 rd   +    16,552,996 wr)
==66518== LL misses:             29,842  (        3,973 rd   +        25,869 wr)
==66518== LL miss rate:             0.0% (          0.0%     +           0.0%   )
==66518==
==66518== Branches:       2,500,403,877  (2,410,401,878 cond +    90,001,999 ind)
==66518== Mispredicts:      168,481,222  (  168,480,870 cond +           352 ind)
==66518== Mispred rate:             6.7% (          7.0%     +           0.0%   )
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/homework_new$
```

以上是sort_p的执行结果

对比执行sort_a

```
sort_a          : Elapsed execution time: 0.507425 sec
sort_a repeated : Elapsed execution time: 0.508748 sec

Running test #1...
 --> test_zero_element at line 245: PASS

Running test #2...
 --> test_one_element at line 266: PASS
Done testing.
==66409==
==66409== I   refs:      24,679,756,372
==66409== I1  misses:             1,611
==66409== LLi misses:             1,517
==66409== I1  miss rate:           0.00%
==66409== LLi miss rate:           0.00%
==66409==
==66409== D   refs:       9,259,541,348  (5,643,703,234 rd   + 3,615,838,114 wr)
==66409== D1  misses:        64,072,064  (   33,465,920 rd   +    30,606,144 wr)
==66409== LLd misses:            28,338  (        2,468 rd   +        25,870 wr)
==66409== D1  miss rate:            0.7% (          0.6%     +           0.8%   )
==66409== LLd miss rate:            0.0% (          0.0%     +           0.0%   )
==66409==
==66409== LL refs:           64,073,675  (   33,467,531 rd   +    30,606,144 wr)
==66409== LL misses:             29,855  (        3,985 rd   +        25,870 wr)
==66409== LL miss rate:             0.0% (          0.0%     +           0.0%   )
==66409==
==66409== Branches:       4,071,679,164  (3,901,676,307 cond +   170,002,857 ind)
==66409== Mispredicts:      308,988,100  (  308,987,744 cond +           356 ind)
==66409== Mispred rate:             7.6% (          7.9%     +           0.0%   )
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/homework_new$ make DEBUG=0
```

可以看到运行时间变快了一点，这是因为每次通过下标访问数组，如a[i]需要进行乘法计算i*4（一个int的大小），而使用指针的情况下pointer++只需要执行一次加法计算

write up 5

> Explain what sorting algorithm you used and how you chose the number of elements to be sorted in the base case. Report on the performance differences you observed.

base case为开始结束递归的最小的数组长度,默认的是要分到1后才会开始归并,这里尝试设置大一点看看运行效果

在sort_c中调用内置的isort插入排序进行测试

```
extern void isort(data_t* start,data_t* end);
```

```
if (p +16< r) {
    int q = (p + r) / 2;
    sort_c(A, p, q);
    sort_c(A, q + 1, r);
    merge_c(A, p, q, r);
  }else{
    isort(A+p,A+r);
  }
```

base case为16

```
Running test #0...
Generating random array of 100000 elements
Arrays are sorted: yes
 --> test_correctness at line 217: PASS
sort_c          : Elapsed execution time: 0.095891 sec
```

```
Arrays are sorted: yes
 --> test_correctness at line 217: PASS
sort_c          : Elapsed execution time: 0.184742 sec

Running test #1...
 --> test_zero_element at line 245: PASS

Running test #2...
 --> test_one_element at line 266: PASS
Done testing.
==67359==
==67359== I   refs:      5,174,035,664
==67359== I1  misses:            1,622
==67359== LLi misses:            1,528
==67359== I1  miss rate:          0.00%
==67359== LLi miss rate:          0.00%
==67359==
==67359== D   refs:      1,703,154,940  (1,030,975,652 rd   + 672,179,288 wr)
==67359== D1  misses:       33,861,683  (   17,348,960 rd   +  16,512,723 wr)
==67359== LLd misses:           28,339  (        2,472 rd   +      25,867 wr)
==67359== D1  miss rate:           2.0% (          1.7%     +         2.5%  )
==67359== LLd miss rate:           0.0% (          0.0%     +         0.0%  )
==67359==
==67359== LL refs:          33,863,305  (   17,350,582 rd   +  16,512,723 wr)
==67359== LL misses:            29,867  (        4,000 rd   +      25,867 wr)
==67359== LL miss rate:            0.0% (          0.0%     +         0.0%  )
==67359==
==67359== Branches:      1,006,462,159  (  989,906,560 cond +  16,555,599 ind)
==67359== Mispredicts:      93,503,781  (   93,503,429 cond +         352 ind)
==67359== Mispred rate:            9.3% (          9.4%     +         0.0%   )
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/homework_new$
```

base case为32

```
==67426==
--67426-- warning: L3 cache found, using its data for the LL simulation.

Running test #0...
Generating random array of 100000 elements
Arrays are sorted: yes
 --> test_correctness at line 217: PASS
sort_c          : Elapsed execution time: 0.081380 sec
```

```
Arrays are sorted: yes
 --> test_correctness at line 217: PASS
sort_c          : Elapsed execution time: 0.167607 sec

Running test #1...
 --> test_zero_element at line 245: PASS

Running test #2...
 --> test_one_element at line 266: PASS
Done testing.
==67426==
==67426== I   refs:      5,172,411,702
==67426== I1  misses:            1,622
==67426== LLi misses:            1,528
==67426== I1  miss rate:          0.00%
==67426== LLi miss rate:          0.00%
==67426==
==67426== D   refs:      1,664,315,176  (  988,949,373 rd   + 675,365,803 wr)
==67426== D1  misses:       33,846,903  (   17,347,635 rd   +  16,499,268 wr)
==67426== LLd misses:           28,337  (        2,472 rd   +      25,865 wr)
==67426== D1  miss rate:           2.0% (          1.8%     +         2.4%   )
==67426== LLd miss rate:           0.0% (          0.0%     +         0.0%   )
==67426==
==67426== LL refs:          33,848,525  (   17,349,257 rd   +  16,499,268 wr)
==67426== LL misses:            29,865  (        4,000 rd   +      25,865 wr)
==67426== LL miss rate:            0.0% (          0.0%     +         0.0%   )
==67426==
==67426== Branches:      1,091,180,840  (1,077,902,041 cond +  13,278,799 ind)
==67426== Mispredicts:      85,443,667  (   85,443,315 cond +         352 ind)
==67426== Mispred rate:             7.8% (          7.9%     +         0.0%   )
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/homework_new$
```

base case为64

```
--67492-- warning: L3 cache found, using its data for the LL simulation.

Running test #0...
Generating random array of 100000 elements
Arrays are sorted: yes
 --> test_correctness at line 217: PASS
sort_c          : Elapsed execution time: 0.087393 sec
```

```
Arrays are sorted: yes
 --> test_correctness at line 217: PASS
sort_c          : Elapsed execution time: 0.191518 sec

Running test #1...
 --> test_zero_element at line 245: PASS

Running test #2...
 --> test_one_element at line 266: PASS
Done testing.
==67492==
==67492== I   refs:      6,027,177,234
==67492== I1  misses:            1,621
==67492== LLi misses:            1,527
==67492== I1  miss rate:          0.00%
==67492== LLi miss rate:          0.00%
==67492==
==67492== D   refs:      1,889,267,732  (1,085,190,596 rd   + 804,077,136 wr)
==67492== D1  misses:       33,821,298  (   17,347,480 rd   +  16,473,818 wr)
==67492== LLd misses:           28,333  (        2,472 rd   +      25,861 wr)
==67492== D1  miss rate:           1.8% (          1.6%     +         2.0%   )
==67492== LLd miss rate:           0.0% (          0.0%     +         0.0%   )
==67492==
==67492== LL refs:          33,822,919  (   17,349,101 rd   +  16,473,818 wr)
==67492== LL misses:            29,860  (        3,999 rd   +      25,861 wr)
==67492== LL miss rate:            0.0% (          0.0%     +         0.0%   )
==67492==
==67492== Branches:      1,388,223,042  (1,376,582,643 cond +  11,640,399 ind)
==67492== Mispredicts:      78,348,857  (   78,348,505 cond +         352 ind)
==67492== Mispred rate:             5.6% (          5.7%     +         0.0%   )
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/homework_new$ make DEBUG=0
```

base case为128

```
Running test #0...
Generating random array of 100000 elements
Arrays are sorted: yes
 --> test_correctness at line 217: PASS
sort_c          : Elapsed execution time: 0.123463 sec
```

```
Arrays are sorted: yes
 --> test_correctness at line 217: PASS
sort_c          : Elapsed execution time: 0.298657 sec

Running test #1...
 --> test_zero_element at line 245: PASS

Running test #2...
 --> test_one_element at line 266: PASS
Done testing.
==67592==
==67592== I   refs:      8,258,419,004
==67592== I1  misses:            1,615
==67592== LLi misses:            1,521
==67592== I1  miss rate:          0.00%
==67592== LLi miss rate:          0.00%
==67592==
==67592== D   refs:      2,515,828,956  (1,385,388,318 rd   + 1,130,440,638 wr)
==67592== D1  misses:       33,745,212  (   17,309,756 rd   +    16,435,456 wr)
==67592== LLd misses:           28,327  (        2,472 rd   +        25,855 wr)
==67592== D1  miss rate:           1.3% (          1.2%     +           1.5%   )
==67592== LLd miss rate:           0.0% (          0.0%     +           0.0%   )
==67592==
==67592== LL refs:          33,746,827  (   17,311,371 rd   +    16,435,456 wr)
==67592== LL misses:            29,848  (        3,993 rd   +        25,855 wr)
==67592== LL miss rate:            0.0% (          0.0%     +           0.0%   )
==67592==
==67592== Branches:      2,065,574,694  (2,054,753,495 cond +    10,821,199 ind)
==67592== Mispredicts:      72,133,953  (   72,133,601 cond +           352 ind)
==67592== Mispred rate:            3.5% (          3.5%     +           0.0%   )
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/homework_new$
```

base case为256

```
Running test #0...
Generating random array of 100000 elements
Arrays are sorted: yes
 --> test_correctness at line 217: PASS
sort_c          : Elapsed execution time: 0.166654 sec
Generating inverted array of 100000 elements
```

```
Arrays are sorted: yes
 --> test_correctness at line 217: PASS
sort_c          : Elapsed execution time: 0.436079 sec

Running test #1...
 --> test_zero_element at line 245: PASS

Running test #2...
 --> test_one_element at line 266: PASS
Done testing.
==67659==
==67659== I   refs:      13,104,047,177
==67659== I1  misses:             1,603
==67659== LLi misses:             1,513
==67659== I1  miss rate:           0.00%
==67659== LLi miss rate:           0.00%
==67659==
==67659== D   refs:       3,892,406,942  (2,062,153,808 rd   + 1,830,253,134 wr)
==67659== D1  misses:        33,593,964  (   17,287,448 rd   +    16,306,516 wr)
==67659== LLd misses:            28,306  (        2,470 rd   +        25,836 wr)
==67659== D1  miss rate:            0.9% (          0.8%     +           0.9%   )
==67659== LLd miss rate:            0.0% (          0.0%     +           0.0%   )
==67659==
==67659== LL refs:           33,595,567  (   17,289,051 rd   +    16,306,516 wr)
==67659== LL misses:             29,819  (        3,983 rd   +        25,836 wr)
==67659== LL miss rate:             0.0% (          0.0%     +           0.0%   )
==67659==
==67659== Branches:       3,482,560,737  (3,472,149,138 cond +    10,411,599 ind)
==67659== Mispredicts:       66,191,173  (   66,190,821 cond +           352 ind)
==67659== Mispred rate:             1.9% (          1.9%     +           0.0%   )
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/homework_new$
```

通过以上对比

base case大小	16	32	64	128	256
random array 排序时间	0.095891	0.081380	0.087393	0.123463	0.166654
inverted array 排序时间	0.184742	0.167607	0.191518	0.298657	0.436079
D1 miss rate	2.0%	2.0%	1.8%	1.3%	0.9%
Mispred rate	9.3%	7.8%	5.6%	3.5%	1.9%

可以看到随着base case的增大，miss rate和mispred rate都在减小。当base case为32和64的时候，random array以及inverted array排序时间都较小。

write up 6

> Explain any difference in performance in your sort_m.c. Can a compiler automatically make this optimization for you and save you all the effort? Why or why not?

定义一个half,将数组下标从p到q都复制到里面。

修改copy_m如下

```
static void merge_m(data_t* A, int p, int q, int r) {
  assert(A);
  assert(p <= q);
  assert((q + 1) <= r);
```

```
  int n1 = q - p + 1;
  //int n2 = r - q;

 // data_t* left = 0, * right = 0;
 // mem_alloc(&left, n1 + 1);
 // mem_alloc(&right, n2 + 1);
  data_t* half=0;
  mem_alloc(&half,n1+1);
  if (half == NULL) {
    mem_free(&half);
    return;
  }

  copy_m(&(A[p]), half, n1);
  //copy_c(&(A[q + 1]), right, n2);
  half[n1] = UINT_MAX;
  //right[n2] = UINT_MAX;

  data_t* l_start=half;
  data_t* r_start=A+q;

  data_t* A0=A+p;
  for (; A0 <= A+r; A0++) {
    if (*l_start <= *r_start) {
      *A0=*l_start;
      l_start++;
    } else {
      *A0=*r_start;
      r_start++;
    }
  }
  mem_free(&half);

}
```

```
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/homework_new$ make DEBUG=0
clang main.c tests.c util.c isort.c sort_a.c sort_c.c sort_i.c sort_p.c sort_m.c sort_f.c -O3 -DNDEBUG -
g -Wall -std=gnu99 -gdwarf-3 -always-inline -lrt -lm  -o sort
clang: warning: argument unused during compilation: '-always-inline' [-Wunused-command-line-argument]
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/homework_new$ valgrind --tool=cachegrind --branch-sim=yes .
/sort 100000 100
==68356== Cachegrind, a cache and branch-prediction profiler
==68356== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==68356== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==68356== Command: ./sort 100000 100
==68356==
--68356-- warning: L3 cache found, using its data for the LL simulation.

Running test #0...
Generating random array of 100000 elements
Arrays are sorted: yes
 --> test_correctness at line 217: PASS
sort_m          : Elapsed execution time: 0.073724 sec
Generating inverted array of 100000 elements
Arrays are sorted: yes
 --> test_correctness at line 217: PASS
sort_m          : Elapsed execution time: 0.169805 sec

Running test #1...
 --> test_zero_element at line 245: PASS

Running test #2...
```

```
Arrays are sorted: yes
 --> test_correctness at line 217: PASS
sort_m          : Elapsed execution time: 0.169805 sec

Running test #1...
 --> test_zero_element at line 245: PASS

Running test #2...
 --> test_one_element at line 266: PASS
Done testing.
==68356==
==68356== I   refs:      5,418,943,190
==68356== I1  misses:            1,593
==68356== LLi misses:            1,509
==68356== I1  miss rate:          0.00%
==68356== LLi miss rate:          0.00%
==68356==
==68356== D   refs:      1,796,007,034  (1,034,900,556 rd   + 761,106,478 wr)
==68356== D1  misses:       23,206,080  (   10,560,248 rd   +  12,645,832 wr)
==68356== LLd misses:           22,054  (        2,472 rd   +      19,582 wr)
==68356== D1  miss rate:           1.3% (          1.0%     +         1.7%  )
==68356== LLd miss rate:           0.0% (          0.0%     +         0.0%  )
==68356==
==68356== LL refs:          23,207,673  (   10,561,841 rd   +  12,645,832 wr)
==68356== LL misses:            23,563  (        3,981 rd   +      19,582 wr)
==68356== LL miss rate:            0.0% (          0.0%     +         0.0%  )
==68356==
==68356== Branches:      1,368,408,208  (1,357,587,611 cond +  10,820,597 ind)
==68356== Mispredicts:      22,660,005  (   22,659,656 cond +         349 ind)
==68356== Mispred rate:            1.7% (          1.7%     +         0.0%   )
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/homework_new$
```

同样base case取64，可以看到和writeup5相比，排序时间减小，D1miss rate从1.8%减小到了1.3%,分支mispred rate从5.6%减小到1.7%。

由于性能有较大提升，可以估计本身的O3等级优化并不会自动对这里进行优化。

write up 7

> Report any differences in performance in your sort_f.c, and explain the differences using profiling data.

将原来的sort_f改为一个驱动函数drive

```
static void drive(data_t* A, int p, int r, data_t* buffer) {
  assert(A);
  if (p + 64 < r) {
  int q = (p + r) / 2;
  drive(A, p, q, buffer);
  drive(A, q + 1, r, buffer);
  merge_f(A, p, q, r, buffer);
  }
  else {
  isort(A+p, A+r);
  }
}
```

在sort_f中申请缓冲区并调用drive函数

```
void sort_f(data_t* A, int p, int r) {
  /*assert(A);
  if (p + 256 < r) {
    int q = (p + r) / 2;
    sort_f(A, p, q);
    sort_f(A, q + 1, r);
    merge_f(A, p, q, r);
  }else{
    isort(A+p,A+r);
  }*/
  assert(A);
  data_t* buffer = 0;
  mem_alloc(&buffer, r-p);
  drive(A, p, r, buffer);
  mem_free(&buffer);
}
```

，并重写一个递归实现的归

并排序函数my_sort_f，在sort_f中申请一个缓冲区用于后续merge操作，并最后释放。

merge_f传参中添加改缓冲区

```
static void merge_f(data_t* A, int p, int q, int r,data_t* buffer)
```

```
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/homework_new$ make DEBUG=0
clang main.c tests.c util.c isort.c sort_a.c sort_c.c sort_i.c sort_p.c sort_m.c sort_f.c -O3 -DNDEBUG -
g -Wall -std=gnu99 -gdwarf-3 -always-inline -lrt -lm  -o sort
clang: warning: argument unused during compilation: '-always-inline' [-Wunused-command-line-argument]
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/homework_new$ valgrind --tool=cachegrind --branch-sim=yes .
/sort 100000 100
==68923== Cachegrind, a cache and branch-prediction profiler
==68923== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==68923== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==68923== Command: ./sort 100000 100
==68923==
--68923-- warning: L3 cache found, using its data for the LL simulation.

Running test #0...
Generating random array of 100000 elements
Arrays are sorted: yes
 --> test_correctness at line 217: PASS
sort_f          : Elapsed execution time: 0.091730 sec
Generating inverted array of 100000 elements
Arrays are sorted: yes
 --> test_correctness at line 217: PASS
sort_f          : Elapsed execution time: 0.208107 sec

Running test #1...
 --> test_zero_element at line 245: PASS
```

```
Arrays are sorted: yes
 --> test_correctness at line 217: PASS
sort_f          : Elapsed execution time: 0.208107 sec

Running test #1...
 --> test_zero_element at line 245: PASS

Running test #2...
 --> test_one_element at line 266: PASS
Done testing.
==68923==
==68923== I   refs:      6,145,560,367
==68923== I1  misses:            1,584
==68923== LLi misses:            1,507
==68923== I1  miss rate:          0.00%
==68923== LLi miss rate:          0.00%
==68923==
==68923== D   refs:      1,931,250,511  (1,098,332,476 rd   + 832,918,035 wr)
==68923== D1  misses:       22,783,212  (   10,247,603 rd   +  12,535,609 wr)
==68923== LLd misses:           22,032  (        2,472 rd   +      19,560 wr)
==68923== D1  miss rate:           1.2% (          0.9%     +         1.5%  )
==68923== LLd miss rate:           0.0% (          0.0%     +         0.0%  )
==68923==
==68923== LL refs:          22,784,796  (   10,249,187 rd   +  12,535,609 wr)
==68923== LL misses:            23,539  (        3,979 rd   +      19,560 wr)
==68923== LL miss rate:            0.0% (          0.0%     +         0.0%  )
==68923==
==68923== Branches:      1,568,883,310  (1,558,881,109 cond +  10,002,201 ind)
==68923== Mispredicts:      22,554,790  (   22,554,441 cond +         349 ind)
==68923== Mispred rate:            1.4% (          1.4%     +         0.0%   )
wangwenqing@ubuntu:~/Desktop/MIT6_172F18_hw2/homework_new$
```

对比writeup6,可以看到排序时间进一步减小，D1miss rate从1.3%减小到了1.2%,分支mispred rate从1.7%减小到1.4%。

reference

https://www.kernel.org/doc/html/latest/admin-guide/perf-security.html

http://www.caotama.com/612336.html

https://stackoverflow.com/questions/21284906/perf-couldnt-record-kernel-reference-relocation-symbol

https://askubuntu.com/questions/1171494/how-to-get-perf-fully-working-with-all-features

https://zhuanlan.zhihu.com/p/74425386

内联测试：https://blog.csdn.net/weixin_39703605/article/details/108410698

https://valgrind.org/docs/manual/cg-manual.html