J

**ĐẠI HỌC QUỐC GIA TPHCM**

**TRƯỜNG ĐẠI HỌC BÁCH KHOA**

·▪•🙢🙞🕮🙜🙠•▪·

**BÀI TẬP LỚN KIẾN TRÚC MÁY TÍNH**

**KIẾN TRÚC TẬP LỆNH MIPS**



**Giáo viên: Trần Thanh Bình**

**Lớp: L05**

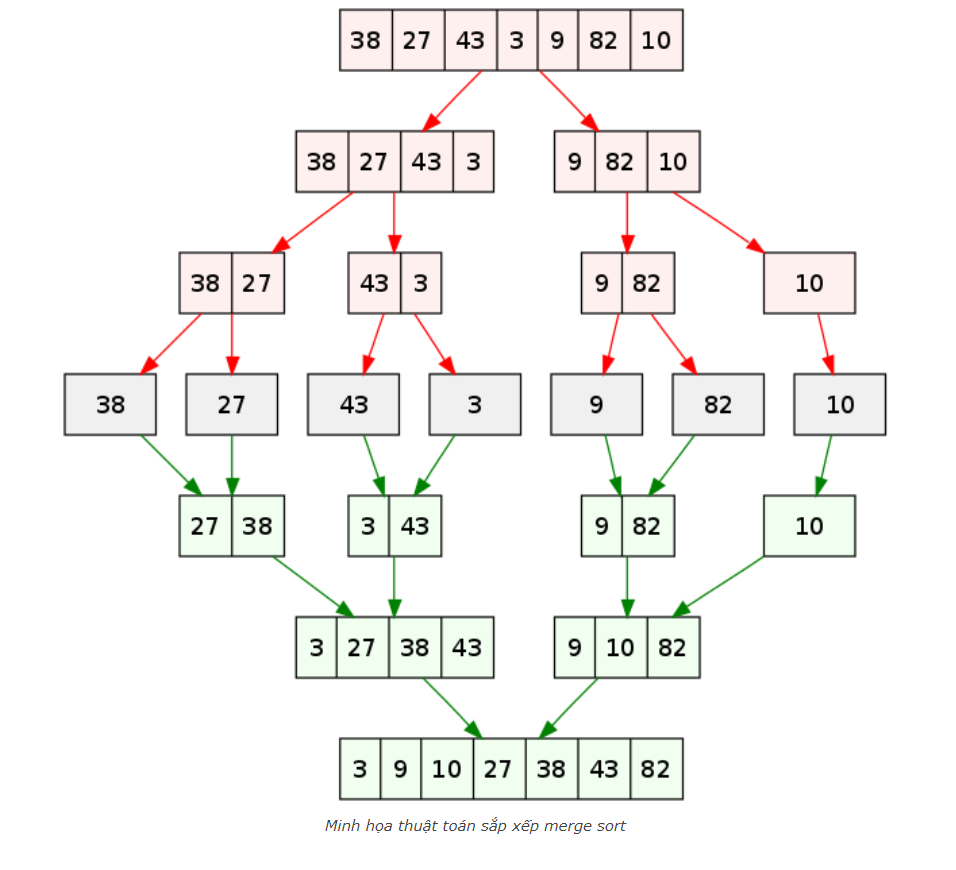
**Vấn đề giải quyết: Problem 6**

|  |  |
| --- | --- |
| Huỳnh Tuấn Đạt | 1913026 |
| Nguyễn Văn Quốc | 1914864 |
| Phan Anh Tú | 1915822 |

**Ngày 4 tháng 12 năm 2020**

1. **Kiến thức cơ sở:**
2. **Ý tưởng của thuật toán Merge Sort:**

Merge sort là một thuật toán chia để trị. Thuật toán này chia mảng cần sắp xếp thành 2 nửa. Tiếp tục lặp lại việc này ở các nửa mảng đã chia. Sau cùng gộp các nửa đó thành mảng đã sắp xếp. Hàm merge() được sử dụng để gộp hai nửa mảng. Hàm merge(arr, l, m, r) là tiến trình quan trọng nhất sẽ gộp hai nửa mảng thành 1 mảng sắp xếp, các nửa mảng là arr[l…m] và arr[m+1…r] sau khi gộp sẽ thành một mảng duy nhất đã sắp xếp.



1. **Đánh giá thuật toán sắp xếp merge sort**

* Độ phức tạp thuật toán:

Trường hợp tốt: O(nlog(n))

Trung bình: O(nlog(n))

Trường hợp xấu: O(nlog(n))

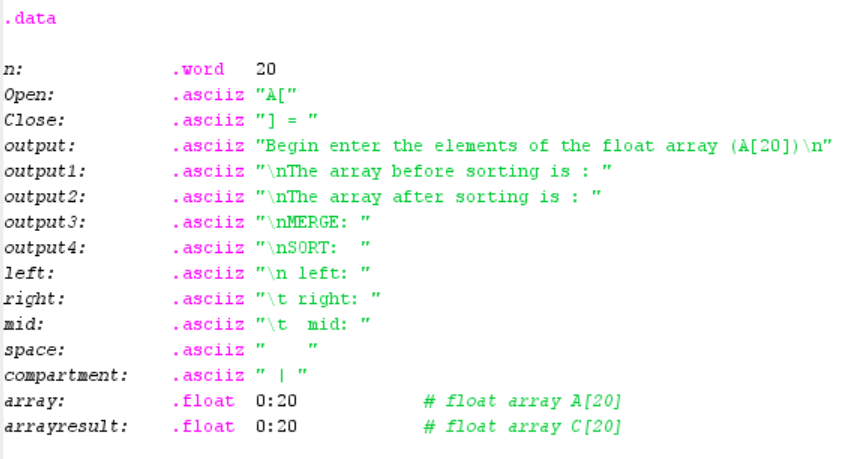
* Không gian bộ nhớ sử dụng: O(n)

1. **Thủ tục MIPS giải quyết vấn đề:**
2. **Loại lệnh đã sử dụng**

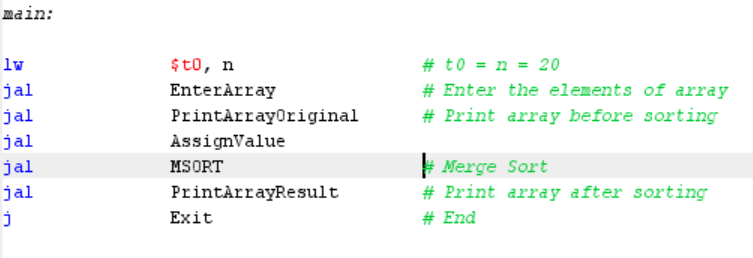
|  |  |  |
| --- | --- | --- |
| Cú pháp | Ảnh hưởng | Miêu tả |
|  | ***Đọc/lưu dữ liệu*** |  |
| lw $s0, 4($s1) | $s0=Mem[$s1+4] | Đọc một từ nhớ ( load word-4 byte) và ghi vào $s0 |
| sw $s0, 4($s1) | Mem[ $t4 + 4 ]=$s0 | Lưu nội dung $s0 vào 4 ô nhớ liên tục bắt đầu từ ô nhớ trong toán hạng bộ nhớ |
| lwc1 $f0, 4($t0) | $f0=Mem[$t0+4] | Load double tại địa chỉ $t0 + 4 |
| swc1 $f0, 4($t4) | Mem[ $t4 + 4 ]=$f0 | Store floating point chính xác đơn vào vùng dữ liệu |
| la $s0, string | $s0=string | Load nội dung string vào $s0 |
| li $s0, a |  | Xét $s0 bằng a. Nếu a 16bits thì mở rộng dấu |
|  | ***Gọi hàm*** |  |
| jr Rs | PC ← Rs | Trở về vị trí thanh ghi Rs trỏ đến |
| jal label | $ra ← PC+4, PC ← label | Gọi hàm label, khi đó $ra nắm vị trí lệnh tiếp theo |
|  | ***Nhảy/rẻ nhánh*** |  |
| j label | PC ← label | Nhảy không điều kiện đến label |
| beq Rs, Rt, label | if (Rs == Rt) PC ← label | Rẽ nhánh đến label nếu Rs == Rt |
| bne Rs, Rt, label | if (Rs != Rt) PC ← label | Rẽ nhánh đến label nếu Rs != Rt |
| bc1f label | if code == 0 then jump to label | Rẽ nhánh |
|  | Số học |  |
| add $s0, $s1, $s2 | $s0=$s1+$s2 | Cộng 2 số nguyên có giá trị ở thanh ghi $s1, $s2 và lưu vào $s0 |
| addi $s0, $s1, 8 | $s0=$s1+8 | Cộng giá trị ở thanh ghi $s1 với 8 rồi lưu vào $s0 |
|  | ***So sánh*** |  |
| slt $s0, $s1, $s2 | if $s1<$s2 then $s0=1 else $s0=0 | So sánh |
| c.lt.s $f2, $f4 | if f2 <f4 then code = 1 else code = 0 | So sánh |
|  | ***Luận lý*** |  |
| sll $s0, $s1, 2 | $s0=$s1\*4 | Dịch trái giá trị thanh ghi nguồn 1 số lượng bit được chỉ định ra trong nguồn 2 |
| or $s0, $s1, $s2 | $s0=$s1 or $s2 | Thực hiện or từng cặp bit trong hai thanh ghi nguồn |

1. **Code thực thi**

* Dữ liệu cần dùng:



* Hàm main:



Với:

- t0 = n = 20 là số phần tử trong mảng.

- EnterArray: Hàm dùng để nhập dữ liệu mảng (các phần tử của mảng).

- PrintArrayOriginal: In mảng đã nhập (chưa sắp xếp).

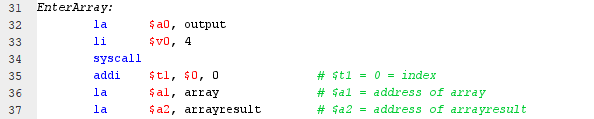
- AssignValue: load lại dữ liệu input (mảng, chỉ số bắt đầu, chỉ số kết thúc, số phần tử) để thực hiện merge sort.

- MSORT: Hàm megre sort (in cụ thể các từng bước).

- PrintArrayResult: In kết quả mảng sau khi đã sort bằng thuật toán merge sort.

- Exit: kết thúc chương trình.

* EnterArray:



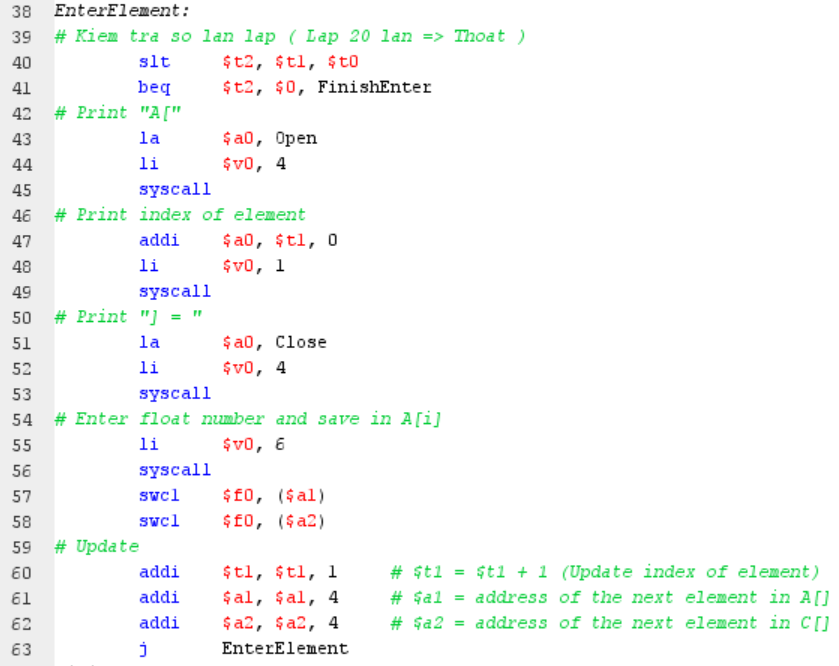
-Dòng 32-34: in ra thông báo "Begin enter the elements of the float array (A[20])" để thông báo việc bắt đầu nhập giá trị các mảng.

-Dòng 35-37: load các giá trị input:

-$t1: vị trí bắt đầu của mảng (=0)

-$a1: mảng cần nhập giá trị các phần tử

-$a2: mảng lưu trữ kết quả( ở đây là kết quả của mảng sau khi nhập và chưa được sort).





-Dòng 40-41:

Kiểm tra xem đã đủ số phần tử chưa. Nếu đã đủ số phần tử thì nhảy đến FinishEnter để đến thúc hàm nhập. Nếu chưa tiếp tục thực hiện việc nhập dữ liệu( các phần tử của mảng).

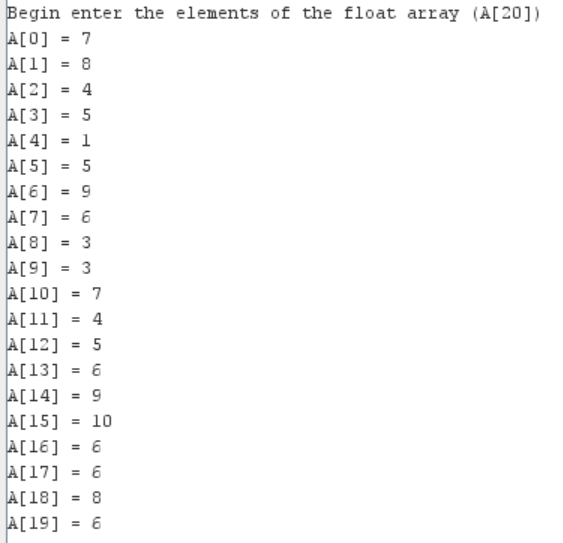
-Dòng 42-58:

In ra màn hình thông báo “A[i]= “. Sau đó nhập từ bàn phím phần tử muốn thêm và lưu vào mảng array($a1)(mảng dữ liệu các phần tử sẽ được sort) và vào mảng arrayresults($s2)( mảng dữ liệu chứa các phần tử chưa được sort)

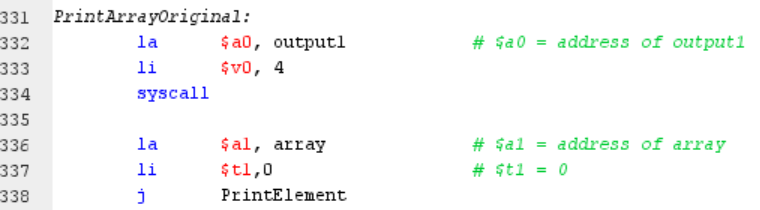
-Dòng 60-63:

Sau khi nhập phần tử thứ i thì tang hệ số đếm lên 1, đồng thời vì mỗi phần từ của mảng chiếm 4 byte nên ta phải cộng địa chỉ array và arrayresults lên 4 để lưu giá trị phần tử mới.

* Ví dụ mảng có các phần tử như sau:

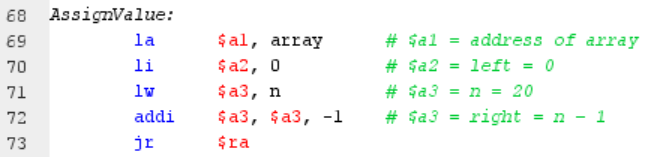


* PrintArrayOriginal



In ra màn hình "The array before sorting is : "

* AssignValue



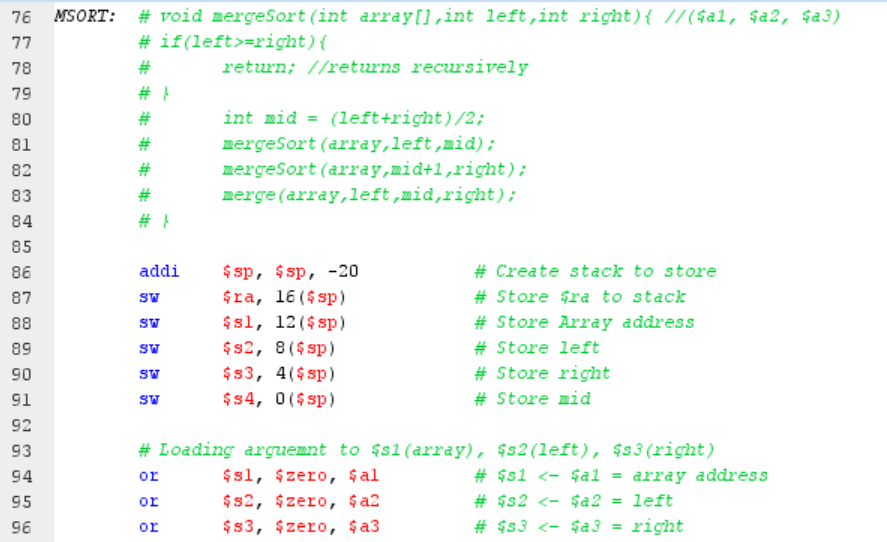
-Mảng cần sắp xếp: $a1

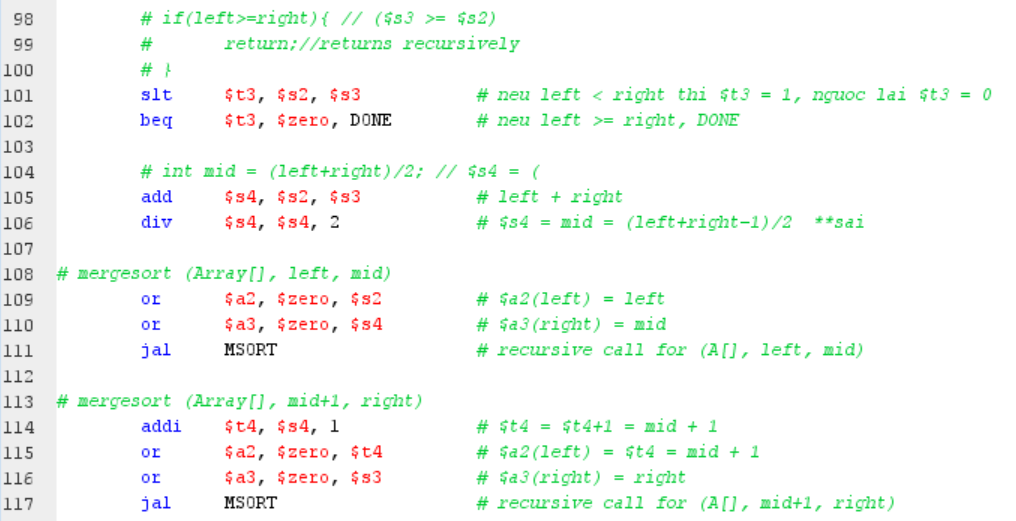
-Chỉ số bắt đầu: $a2 (=0)-Chỉ số kết thúc: $a3 (=20-1=19)

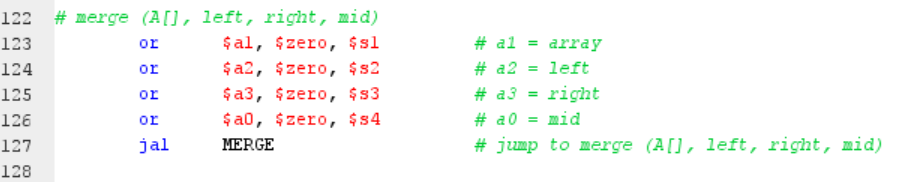
* MSORT:

Mảng gồm 20 phần tử thì merge sort sẽ thực hiện như sau:









-Dòng 89-91:

Lưu trữ giá trị các thanh ghi $ra, $s1, $s2, $s3, $s4 vào stack để khi gọi đệ quy ta có thể lấy lại giá trị lúc trước của các thanh ghi này để sử dụng

-Dòng 94-96: Gán:

+$s1=$a1( địa chỉ của mảng cần sắp xếp)

+$s2=$a2( chỉ số bắt đầu (left))

+$s3=$a3( chỉ số kết thúc (right))

-Dòng 101-111:

Kiểm tra xem nếu chỉ số bắt đầu(left)<= chỉ số kết thúc(right) thì ta thực hiện hàm DONE. Nếu không ta tiếp tục chia nhỏ mảng( với chỉ số bắt đầu là left và chỉ số kết thúc bây giờ là mid+1=(left+right)/2) thành 2 mảng nhỏ hơn. Mảng 1 có chỉ số bắt đầu là left và chỉ số kết thúc là mid( mid lúc này là (left+right)/2 với right=mid trước đó) và mảng thứ 2 chỉ số bắt đầu là mid+1 và chỉ số kết thúc là right.

- Dòng 113-117:

Tương tự như trên, ta thực hiện việc chỉ nhỏ mảng có chỉ số bắt đầu left=mid+1 và chỉ số kết thúc là right

-Dòng 122-127:

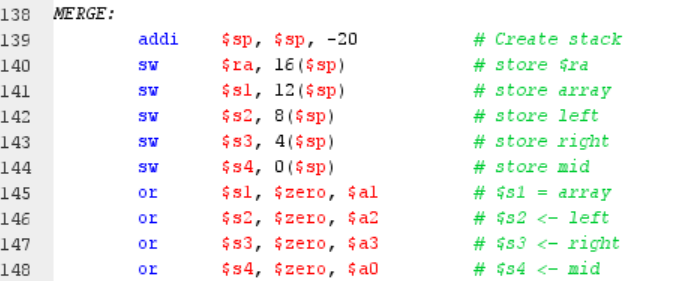
Kết hợp mảng có chỉ số bắt đầu left, chỉ số kết thúc là mid với mảng có chỉ số bắt đầu là mid+1 và chỉ số kết thúc là right để tạo thành mảng hoàn chỉnh.

* DONE:



Sau khi điều kiện left>=right đúng, ta thực hiện việc load $ra các biến mảng, left, right, mid lần lượt ở stack vào các thanh ghi $ra, $s1, $s2, $s2, $s4 để thực hiện việc tính toán.

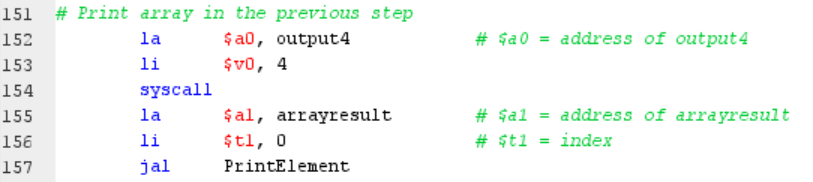
* MERGE:

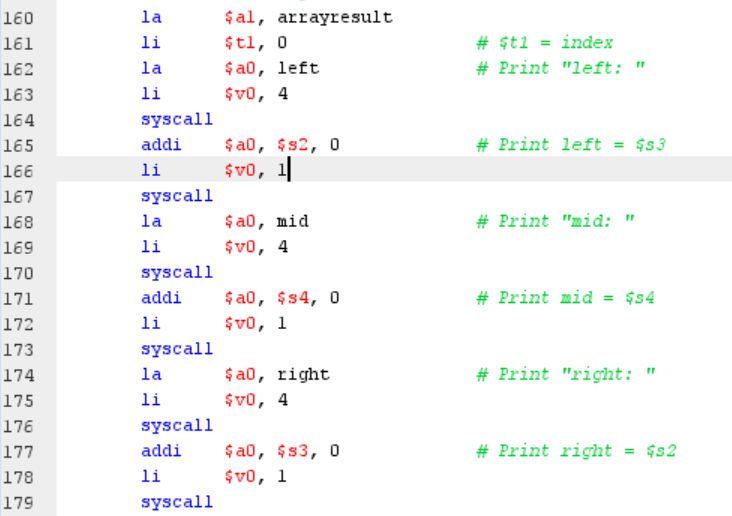


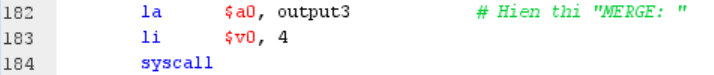
Sau khi có được chia nhỏ các mảng đến mảng con nhỏ hơn, ta thực hiện việc merge các mảng đó lại để tạo ra mảng hoàn chỉnh.

-Dòng 138-148:

Lưu trữ lại giá trị trong các thanh ghi $ra $s1, $s2, $s3, $s4 vào trong stack để trong quá trình hoàn thành lần merge trước đó, ta có thể load lại giá trị của những thanh ghi này để tiếp tục merge các mảng tiếp theo.

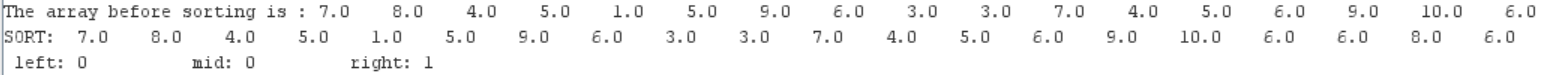






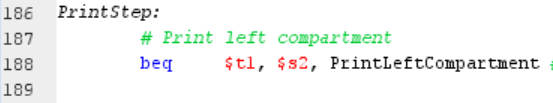
-Dòng 151-184: In mảng chuẩn bị sort và các giá trị left, mid ,right trong lần merge này.

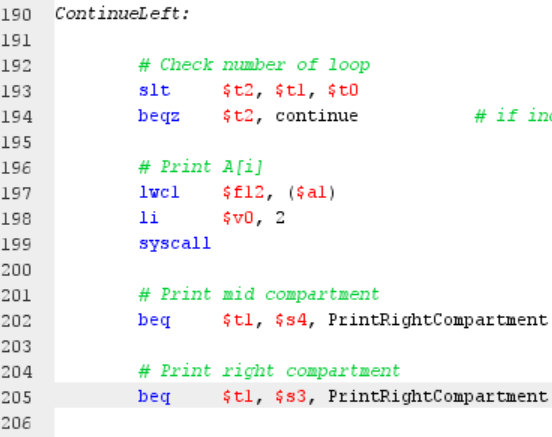
Ví dụ ở lần merge đầu tiên:

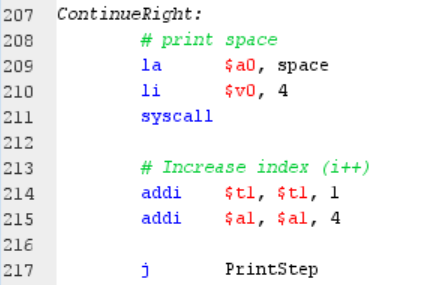


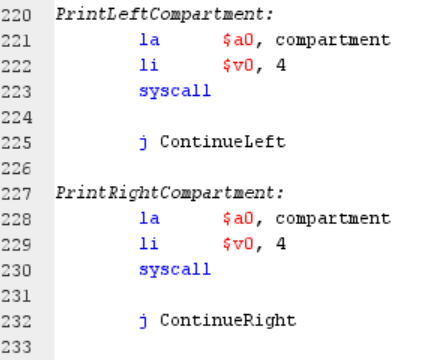
* PrintStep:

Đây là hàm in các bước merge của mảng.









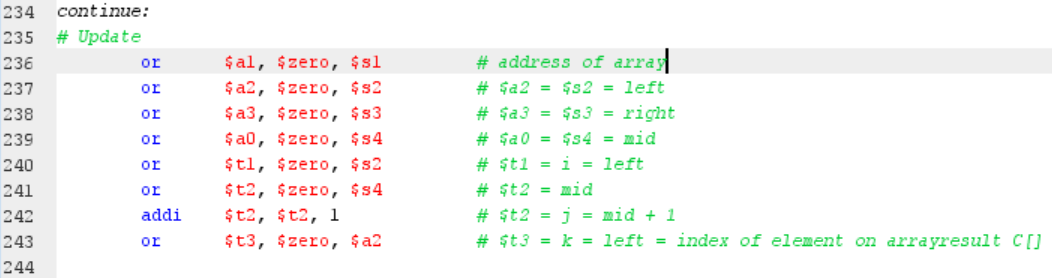
-Dòng 186-233:

Kiểm tra nếu chỉ số đếm($t1) mà bằng chỉ số bắt đầu của mảng thì lúc này ta in dấu ngăn cách đầu tiên để phân định phần tử đầu tiền của mảng cần merge(PrintLeftCompartment). Ở hàm PrintLeftCompartment có dùng lệnh j ContinueLeft. Hàm sẽ trở lại việc so sánh với số phần tử của mảng cần sắp xếp. Nếu đã bằng có nghĩa là ta đã in đủ số phần tử của mảng, ta chuyển đến hàm countinue để thực hiện việc update mảng; nếu không ta tiếp tục kiểm tra chỉ số đếm có bằng mid và right hay không để in các dấu ngăn cách để chỉ ra phần mảng đầu( từ left đến mid) và mảng sau( từ mid+1 đến right).

Kết quả:



* Countine:



-Dòng 236-243: Lưu các biến sau vào các thanh ghi

+Mảng cần sắp xếp: $a1

+Chỉ số bắt đầu(left): $a2

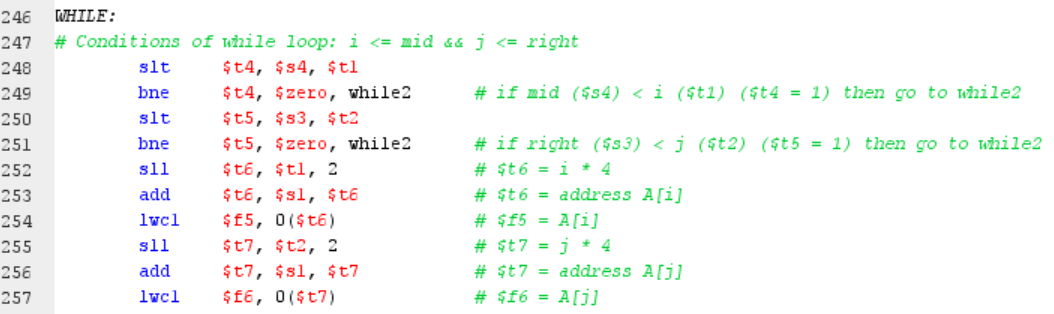
+Chỉ số kết thúc(right): $a3

+Chỉ số mid: $a0

+1 biến đếm i bắt đầu từ left: $t1

+1 biến đếm j bắt đầu từ mid+1: $t2

+1 biến đếm k bắt đầu từ left: $t3



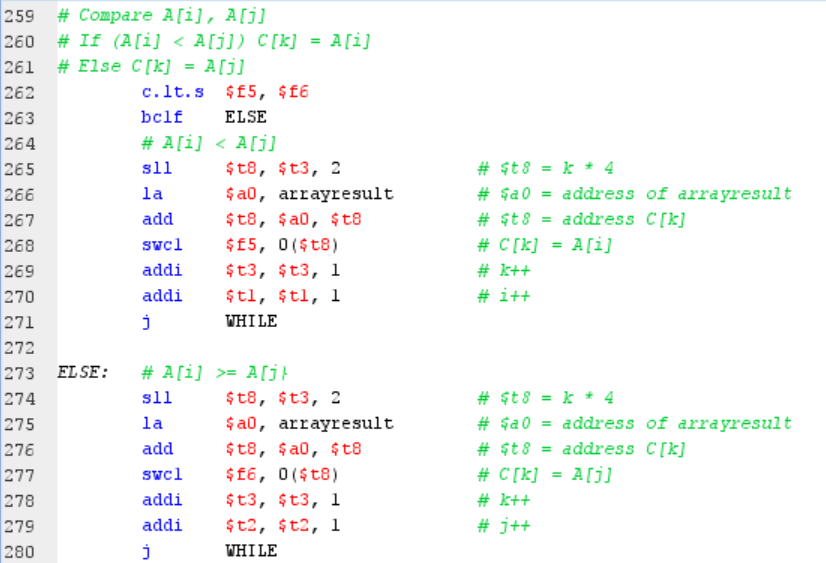
Dòng 248-257:

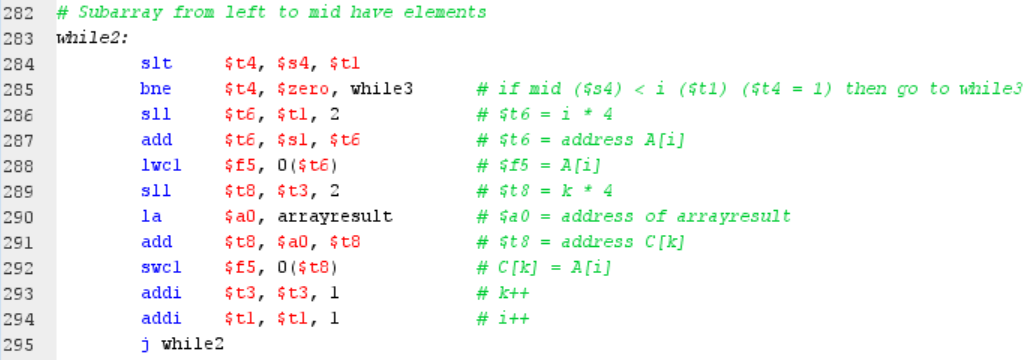
+Kiếm tra biến đếm i có <= mid hay không để xác định xem mảng từ left->mid các phần từ đã được chèn hết vào mảng arrayresults chưa.

+Tương tự như vậy cho biến đếm k và mảng từ mid+1->right.

+Nếu đã chèn hết thì nhảy đến while2

+Nếu chưa đưa phần tử thứ i của mảng từ left->mid vào $f5 và phần tử thứ j của mảng từ mid+1->right vào $f6





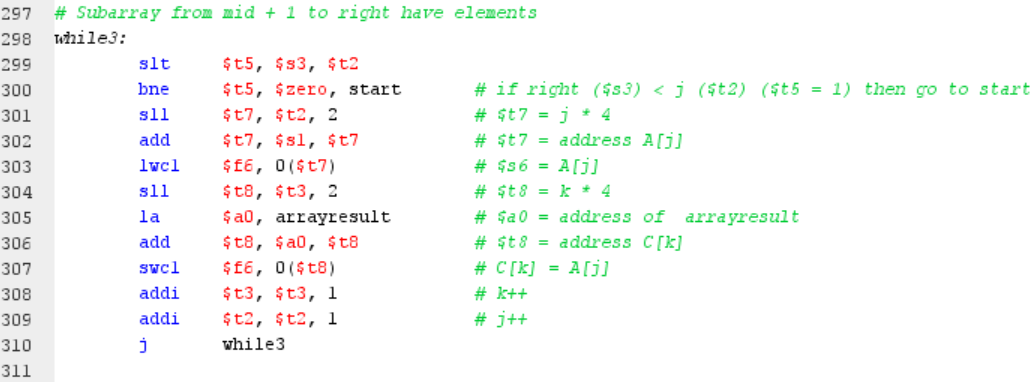
Dòng 260-280:

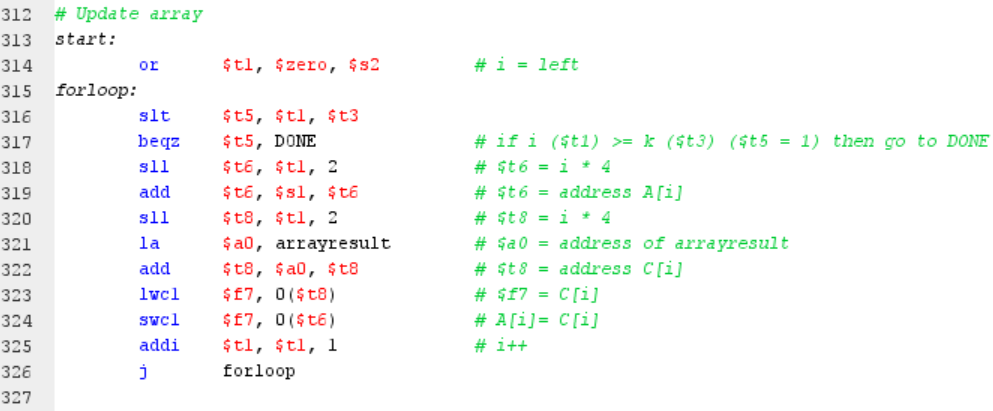
Sau khi đưa 2 phần tử thứ i và thứ j của 2 mảng con đó vào $f5 và $f6 thì bắt đầu cho sánh. Nếu giá trị thanh ghi $f5<$s6( tức là A1[i]<A2[j]) thì đưa giá trị thanh ghi $f5 vào arrayresult và sau đó tăng biến đếm k( dùng để đếm số phần tử của arrayresult) và i; sau đó nhảy về lại WHILE. Ngược lại thì đưa giá trị thanh ghi $f6 vào arrayresult và sau đó tăng biến đếm k( dùng để đếm số phần tử của arrayresult) và j; sau đó nhảy về lại WHILE

Dòng 283-295:

while2: Kiểm tra xem mid < I hay không. Nếu có thì mảng từ left->mid đã được chèn hết vào arrayresult nên nhảy đến while3.

Nếu không ta thực hiện chèn hết tất cả các phần tử còn lại của mảng rừ từ left->mid vào arrayresult và từng bước nhảy lại while2 để kiểm tra điều kiện.





Dòng 297-311:

While3: Kiểm tra xem right < j hay không. Nếu có thì mảng từ mid+1->right đã được chèn hết vào arrayresult nên nhảy đến start.

Nếu không ta thực hiện chèn hết tất cả các phần tử còn lại của mảng rừ từ mid+1->right vào arrayresult và từng bước nhảy lại while3 để kiểm tra điều kiện.

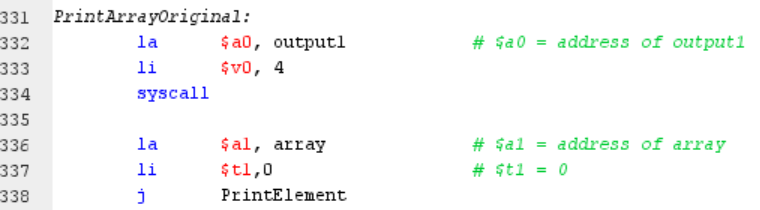
Dòng 312-327:

+Sau khi chèn đầy đủ các phần tử của 2 mảng left->mid và mid+1->right ta sẽ chèn các phần arrayresult vào array để thay đổi vị trí các phần tử ta đã merge sort

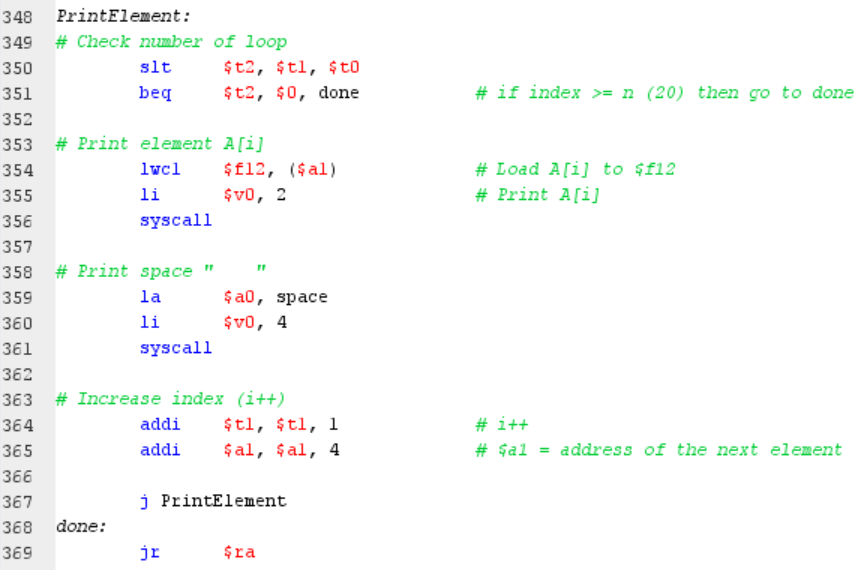
+Gán biến left vào thanh ghi $t1(biến đếm i).

Nếu như i>=k thì có nghĩa là tất cả các phần tử còn lại đã chèn xong và nhảy đến DONE. Nếu không ta chèn các phần của arrayresult bắt đầu từ left đến k vào array và quay lại forloop để kiểm tra điều kiện dừng.

* PrintArrayOriginal



* PrintElement

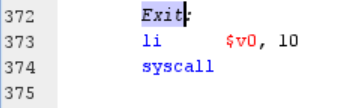


In lần lượt các phần tử của mảng

Kết quả:



* Exit



* Thuật toán được gọi đệ quy cho đến khi tất cả các phần tử được sắp xếp theo thứ tự tăng dần thì dừng lại.

1. **Số lệnh và thời gian chạy**

**Chú thích:**

-CPU time: thời gian xử lý (không tính thời gian giao tiếp I/O, thời gian chờ ...)

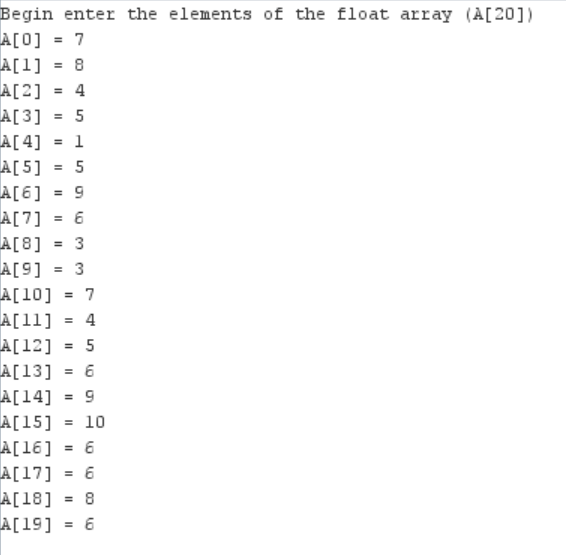
-Clock frequency (rate): Số chu kỳ trên một giây hay còn gọi là tần số, ví dụ: 4GHz = trong 1 giây có 4x109 giao động

-Clock cycles: Tổng số chu kỳ thực thi.

-IC: instruction count, tổng số lệnh thực thi.

-CPI: cycle per instruction (số chu kỳ thực thi trên một lệnh)

**Input 1:**



Số lệnh đã thực hiện: 16598

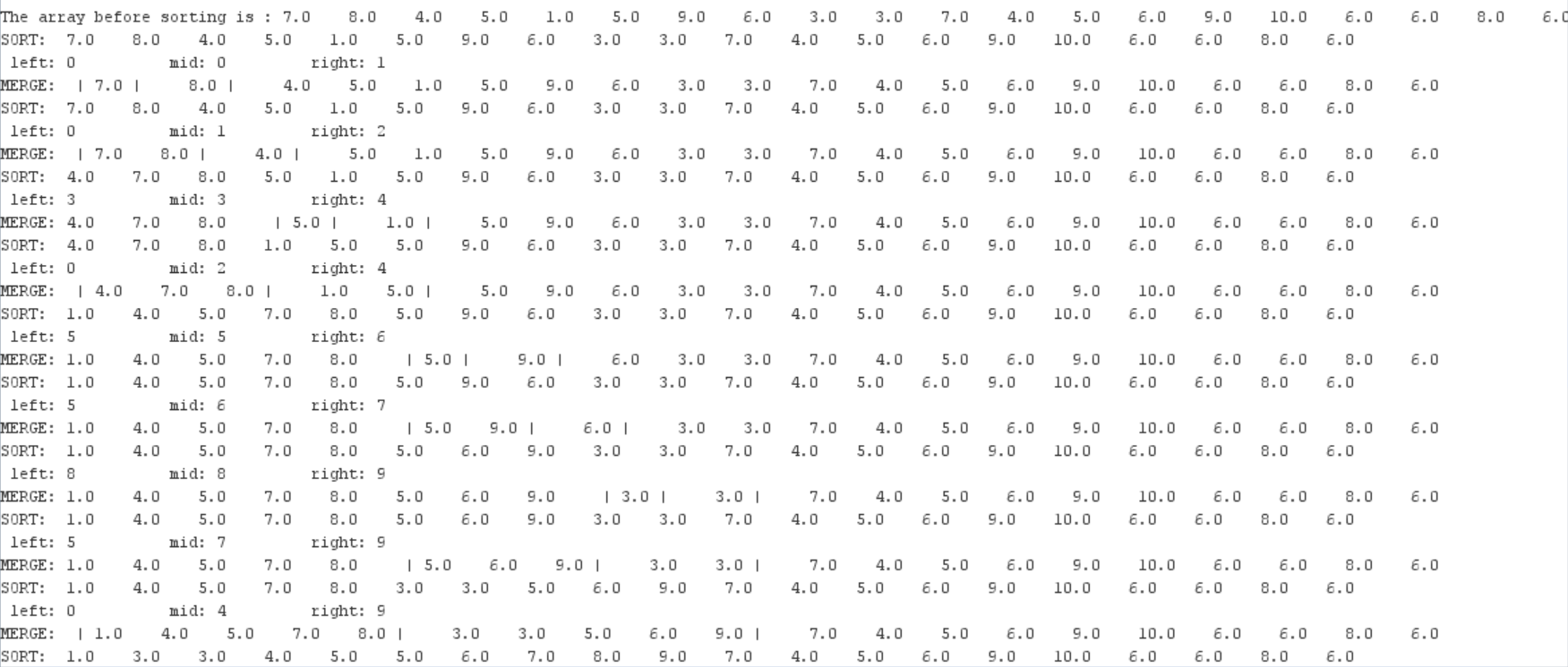


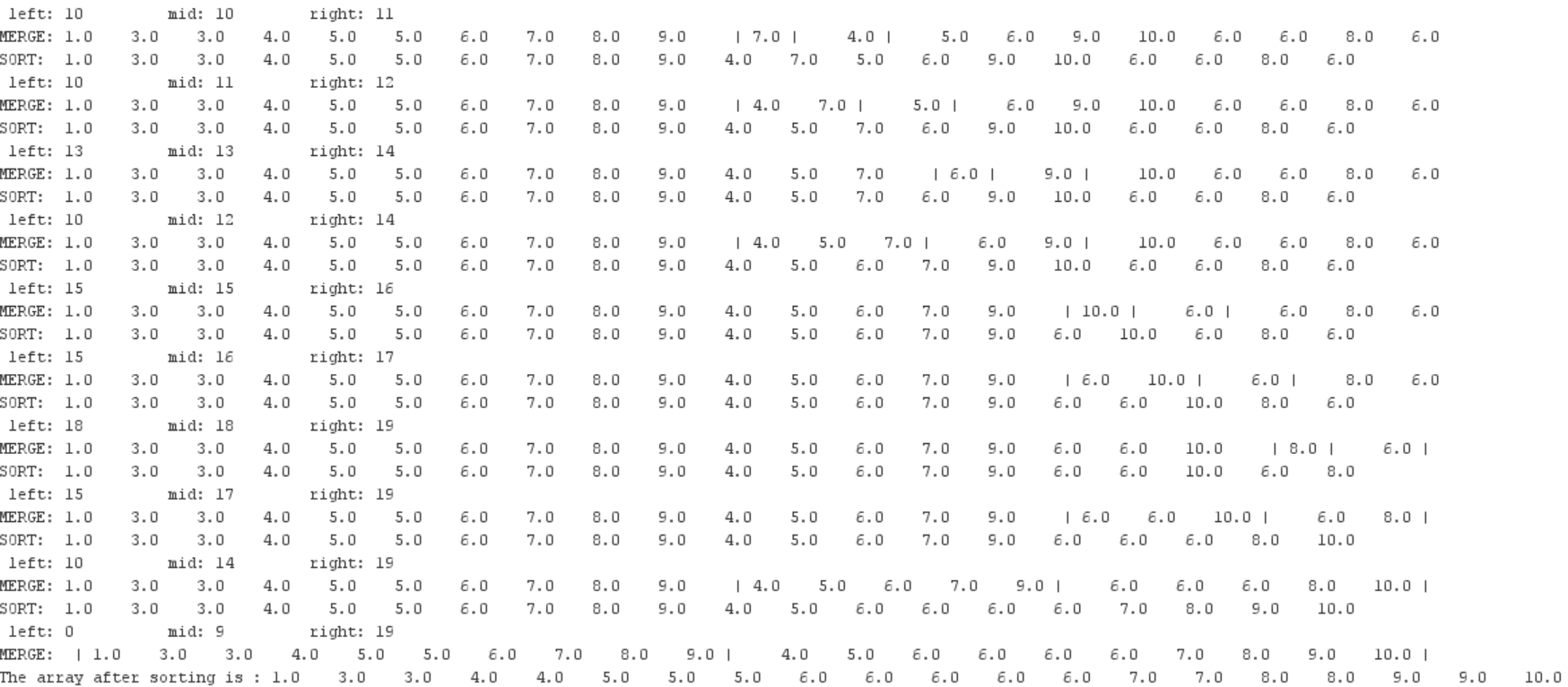
Thời gian thực thi chương trình:

CPU time=IC\*CPI\*clock cycle time=(IC\*CPI)/Clock rate

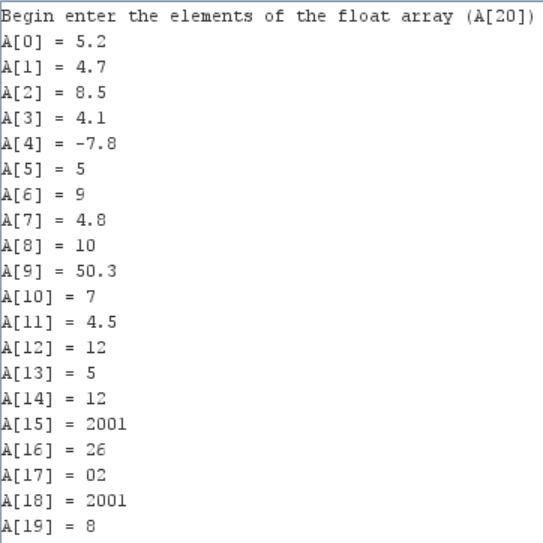
* CPU time= (16598\*1)/2\*10^9=8.3\*10^-6(s)

**Output 1**:

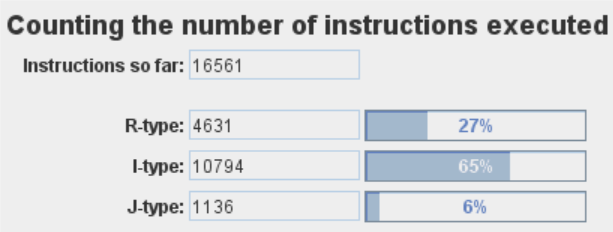




**Input 2:**



Số lệnh đã thực hiện: 16561

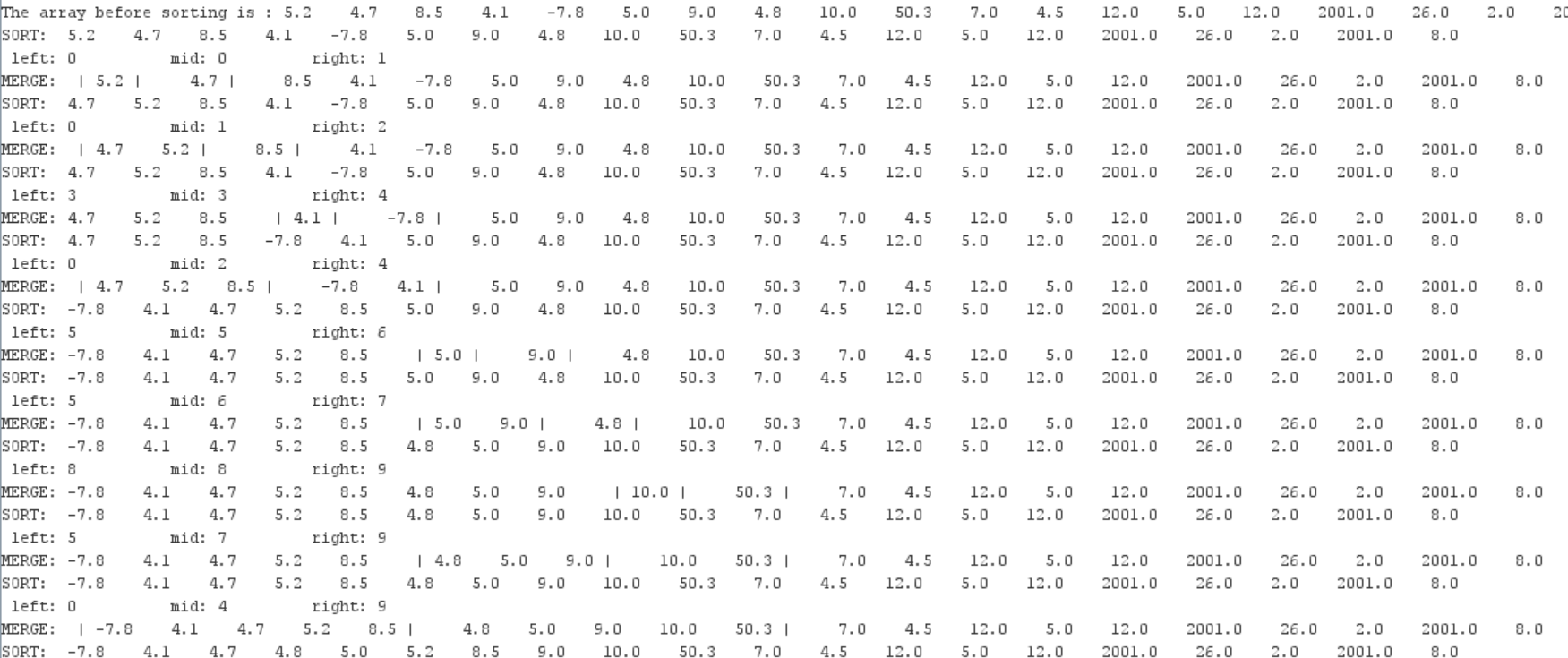


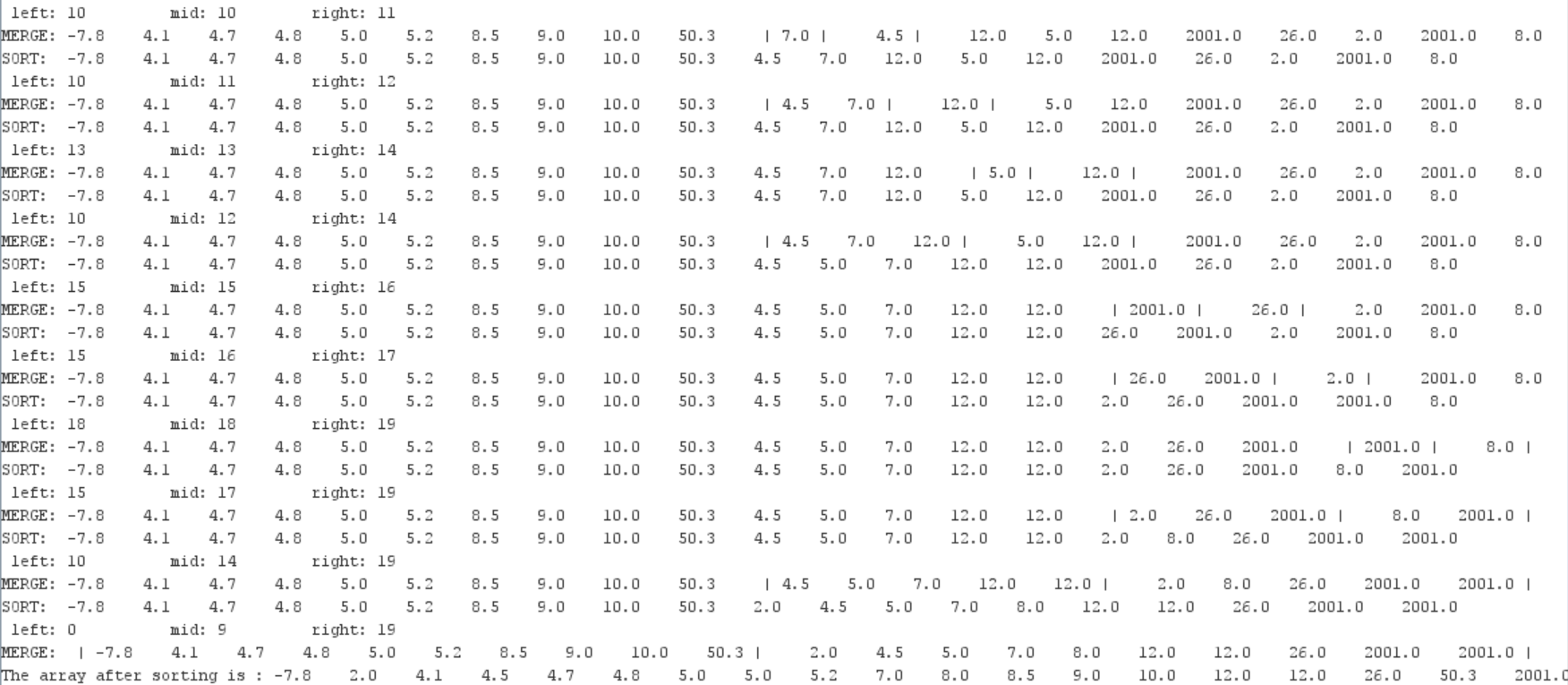
Thời gian thực thi chương trình:

CPU time=IC\*CPI\*clock cycle time=(IC\*CPI)/Clock rate

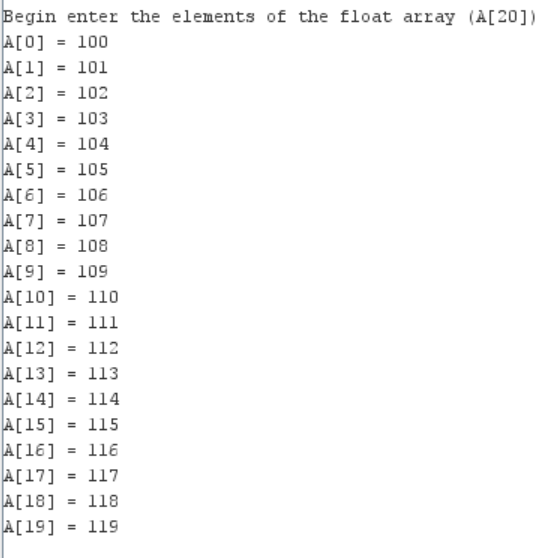
* CPU time= (16561\*1)/2\*10^9=8.28\*10^-6(s)

**Output 2**:

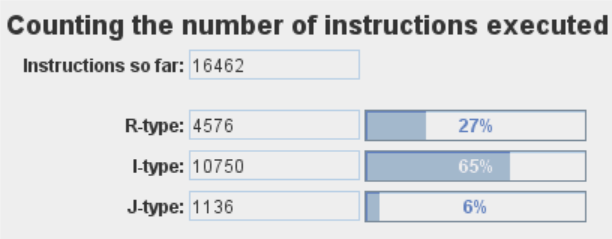




**Input 3:**



Số lệnh thực thi: 16462

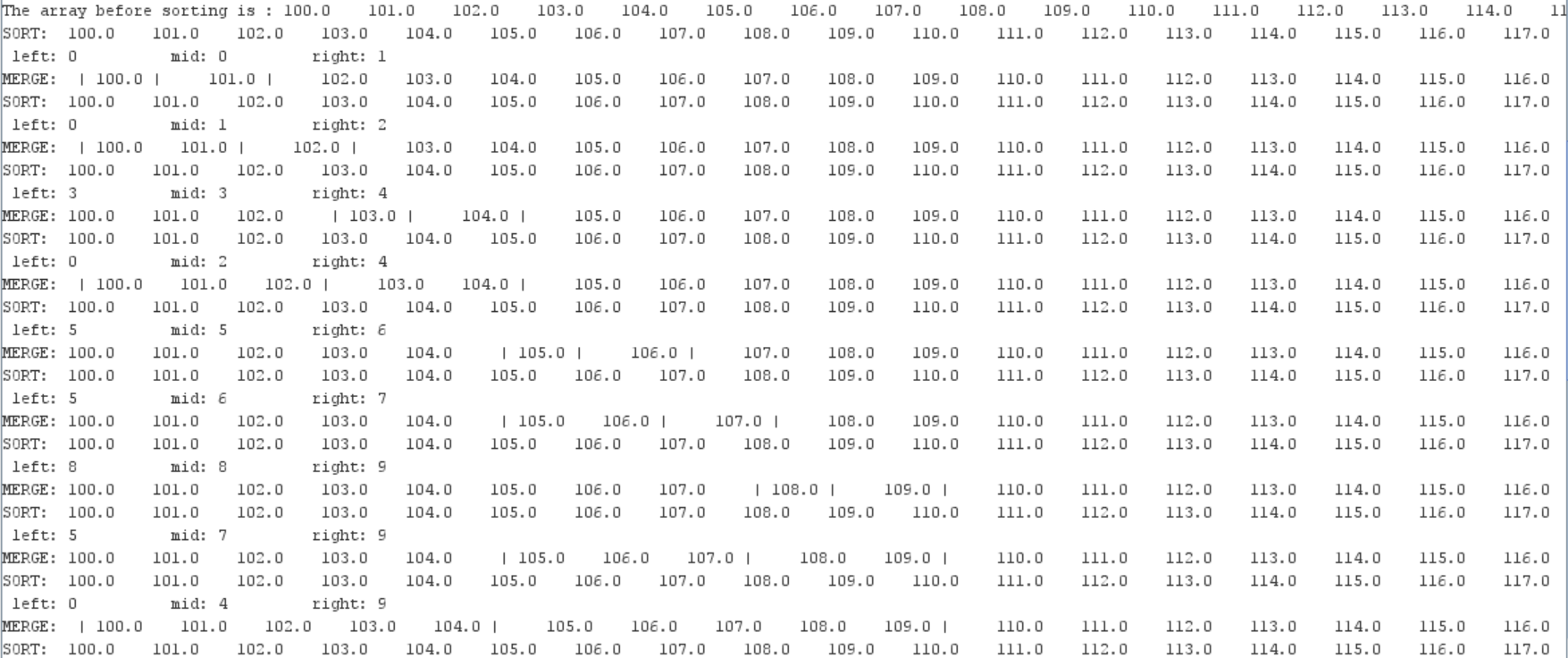


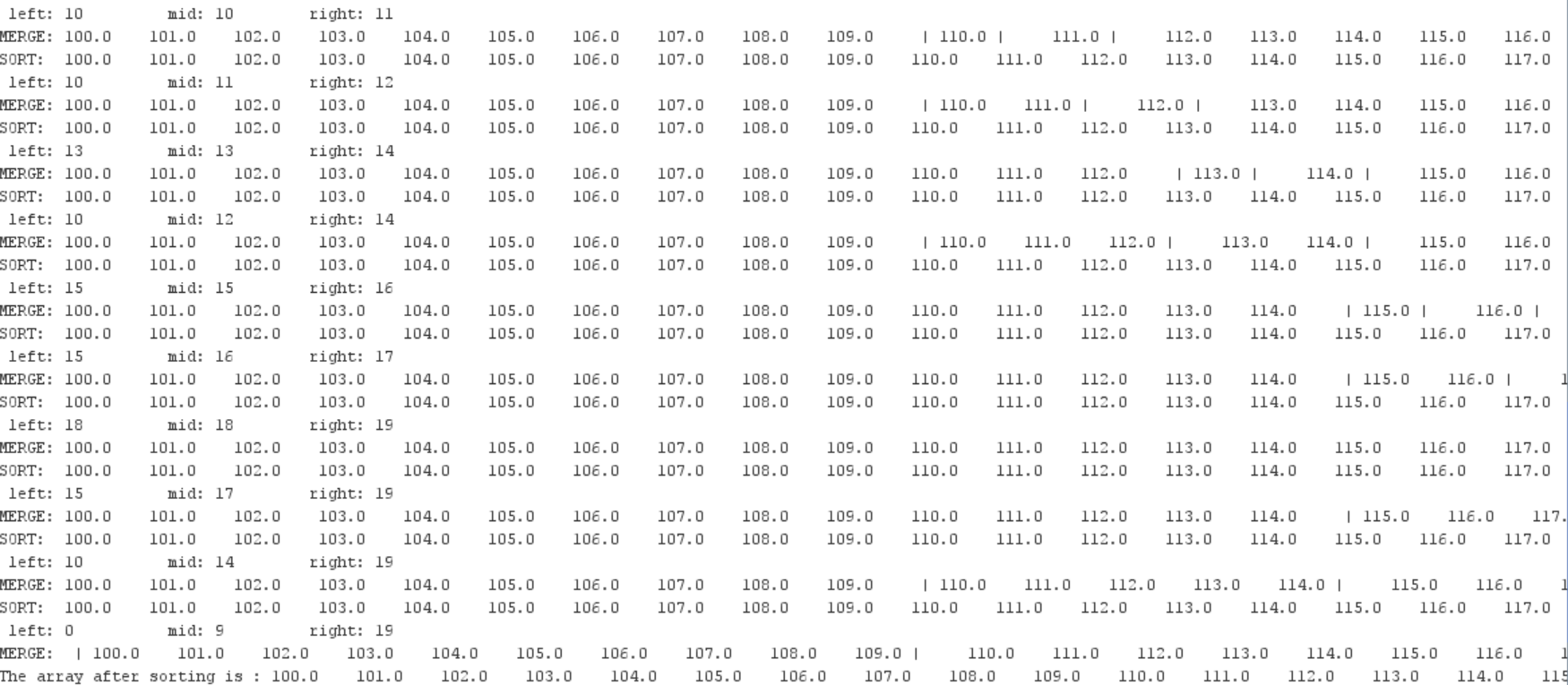
Thời gian thực thi chương trình:

CPU time=IC\*CPI\*clock cycle time=(IC\*CPI)/Clock rate

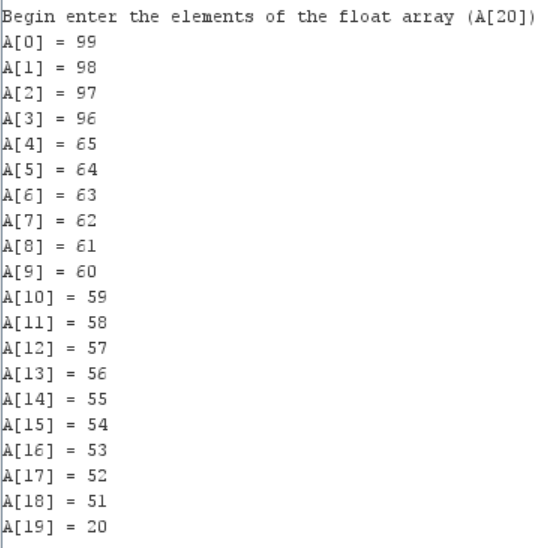
* CPU time= (16462\*1)/2\*10^9=8.23\*10^-6(s)

**Output 3**:

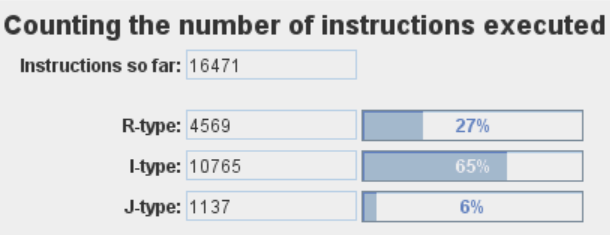




**Input 4:**



Số lệnh thực thi: 16741

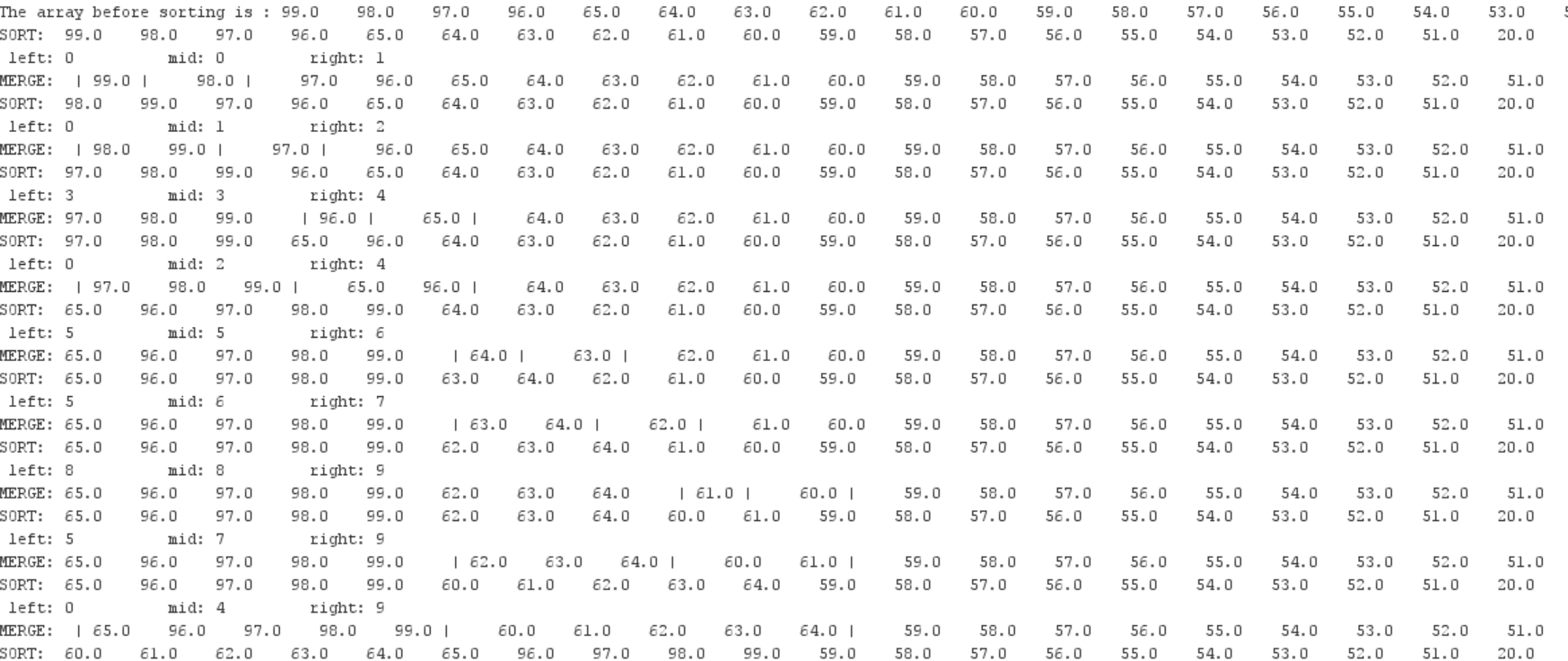


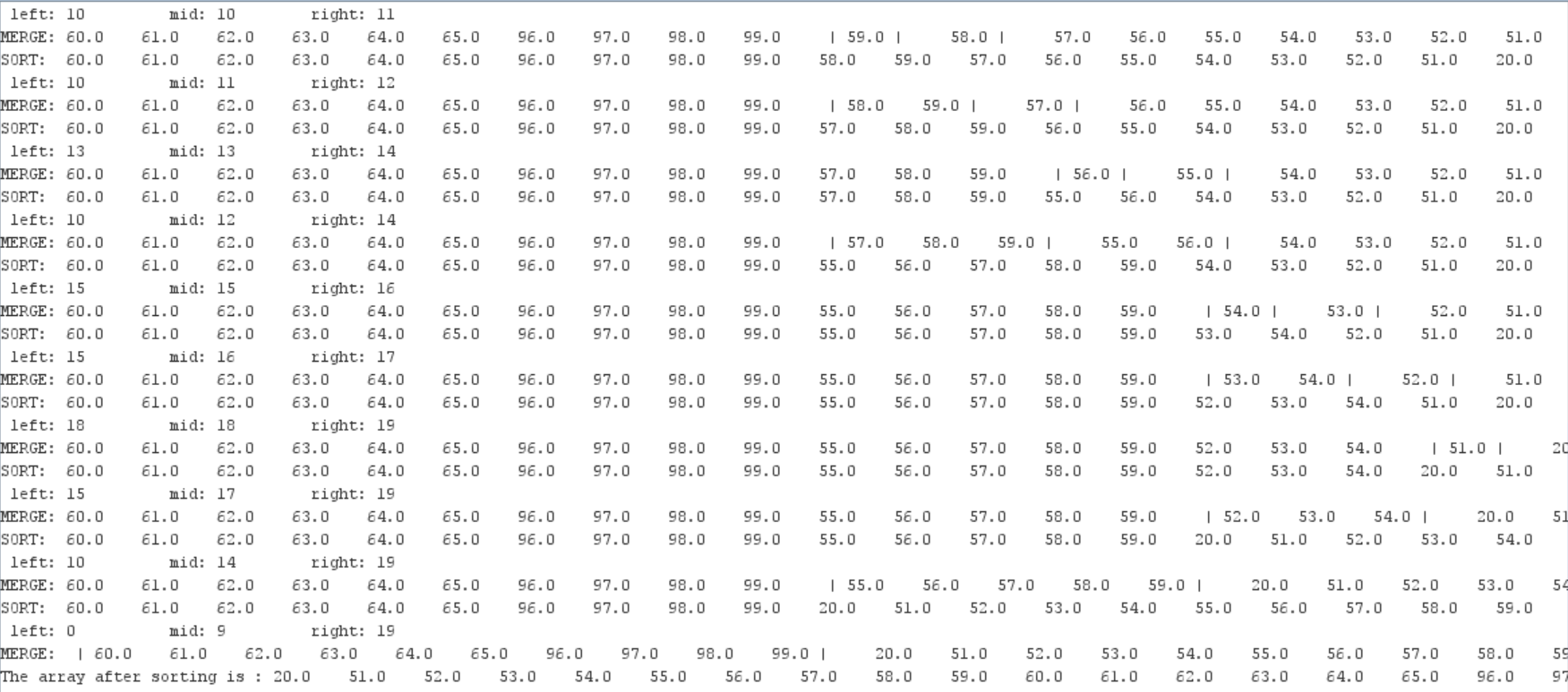
Thời gian thực thi chương trình:

CPU time=IC\*CPI\*clock cycle time=(IC\*CPI)/Clock rate

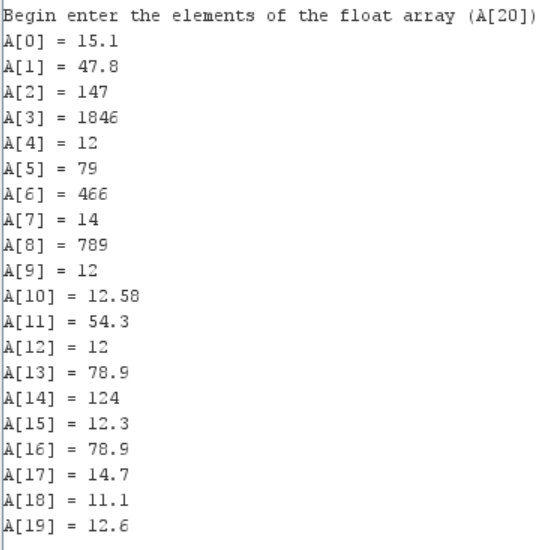
* CPU time= (16741\*1)/2\*10^9=8.37\*10^-6(s)

**Output 4**:

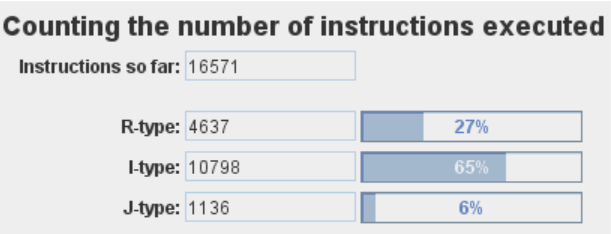




**Input 5:**



Số lệnh thực thi: 16571



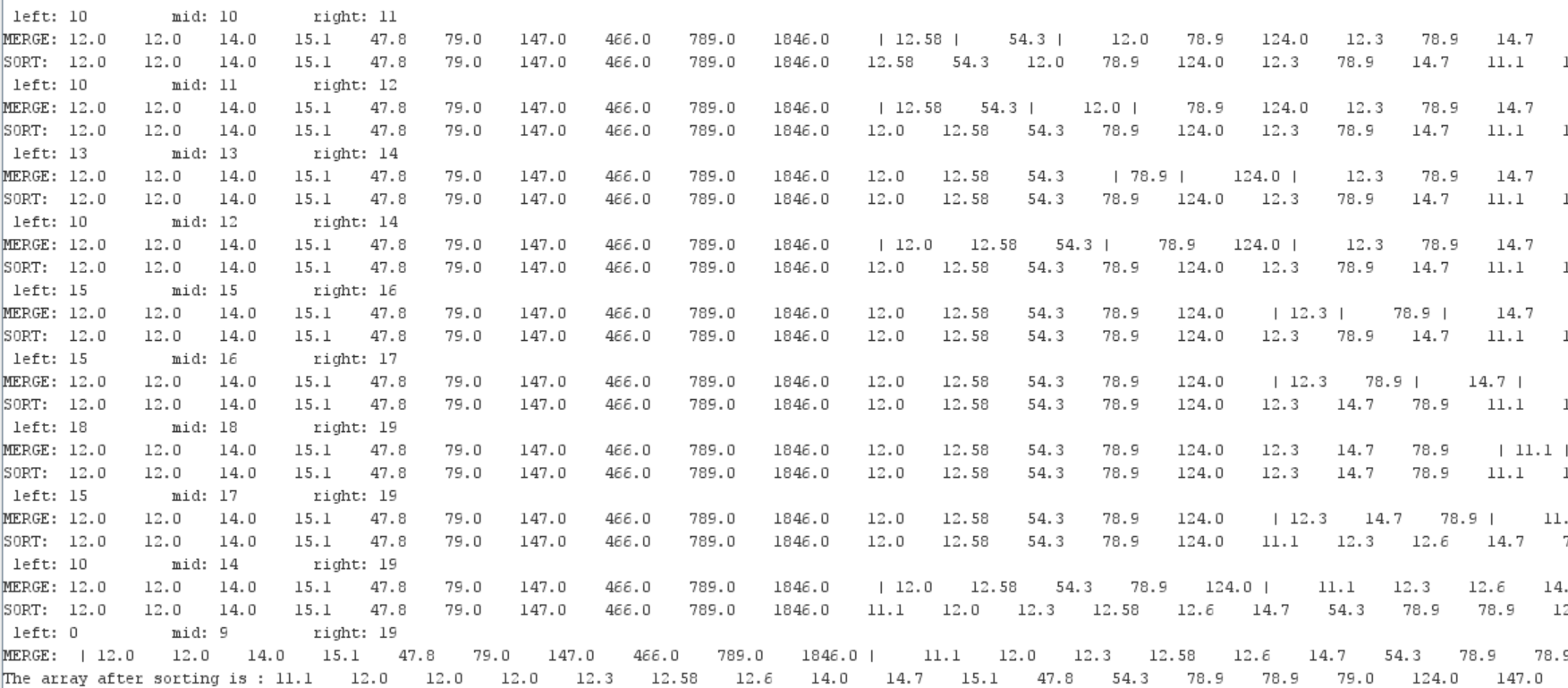
Thời gian thực thi chương trình:

CPU time=IC\*CPI\*clock cycle time=(IC\*CPI)/Clock rate

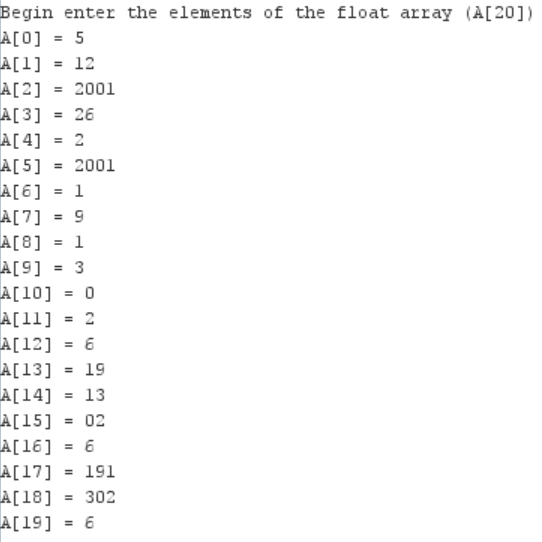
* CPU time= (16571\*1)/2\*10^9=8.29\*10^-6(s)

**Output 5**:

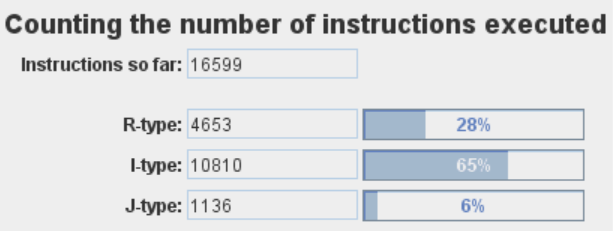




**Input 6:**



Số lệnh thực thi: 16599

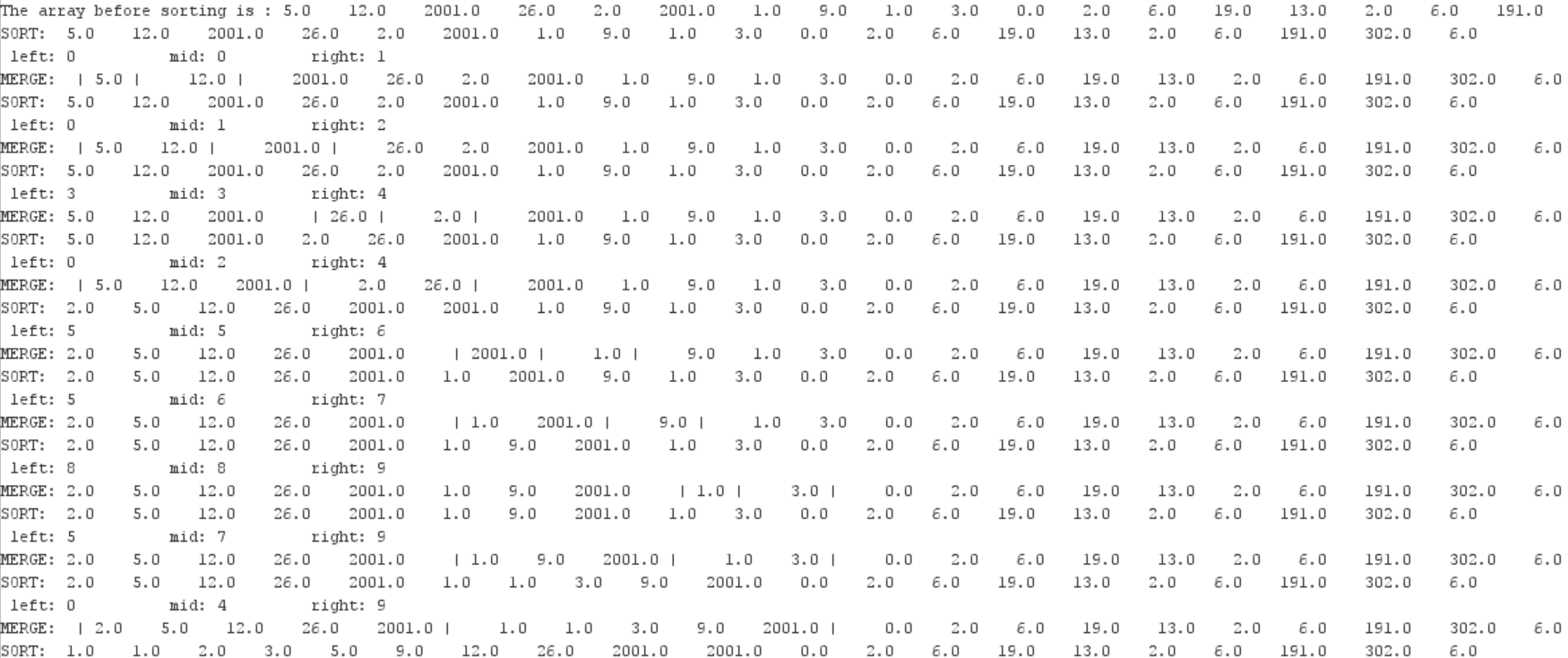


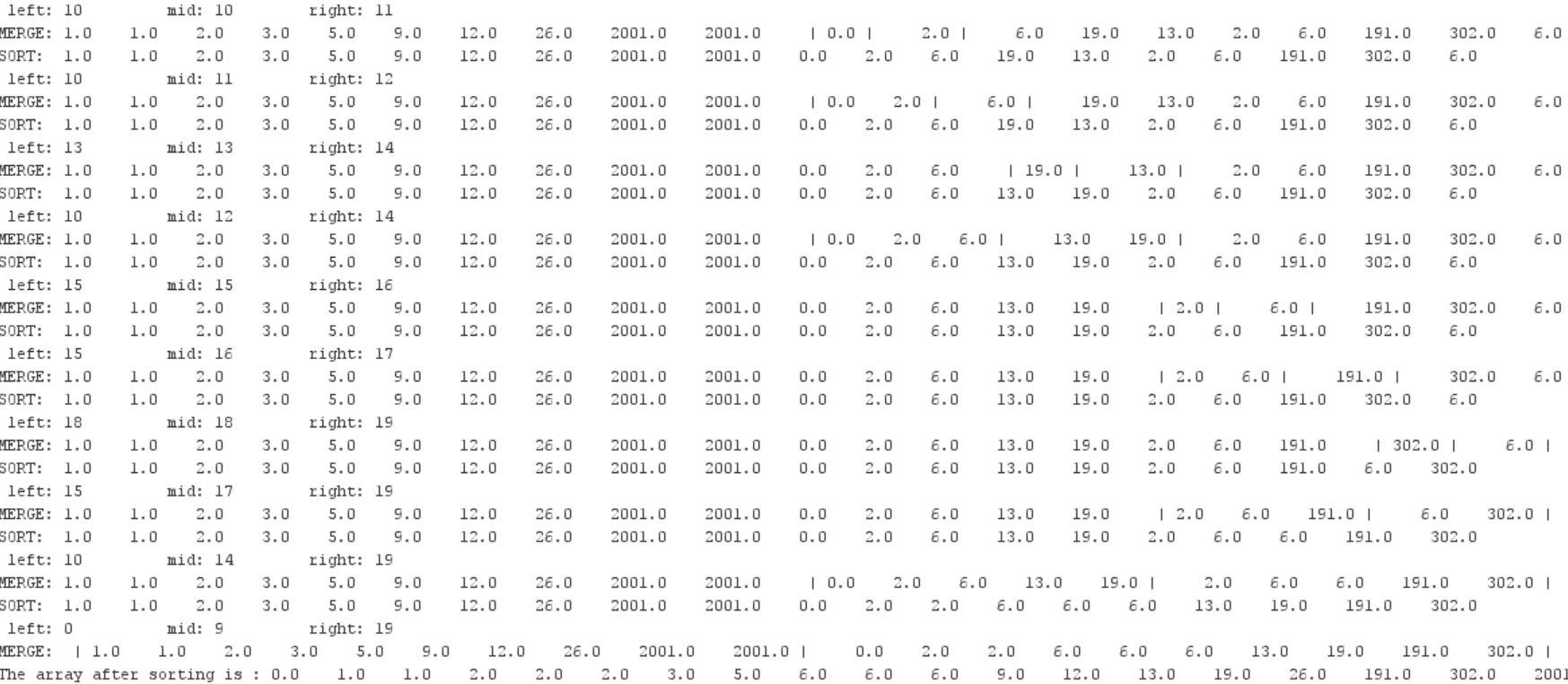
Thời gian thực thi chương trình:

CPU time=IC\*CPI\*clock cycle time=(IC\*CPI)/Clock rate

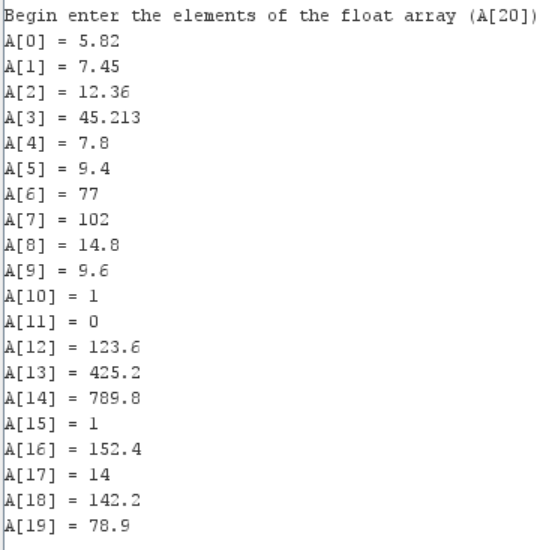
* CPU time= (16599\*1)/2\*10^9=8.3\*10^-6(s)

**Output 6**:

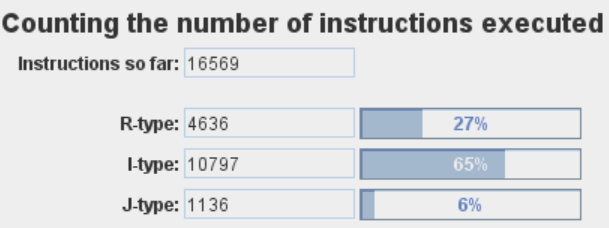




**Input 7:**



Số lệnh thực thi: 16569

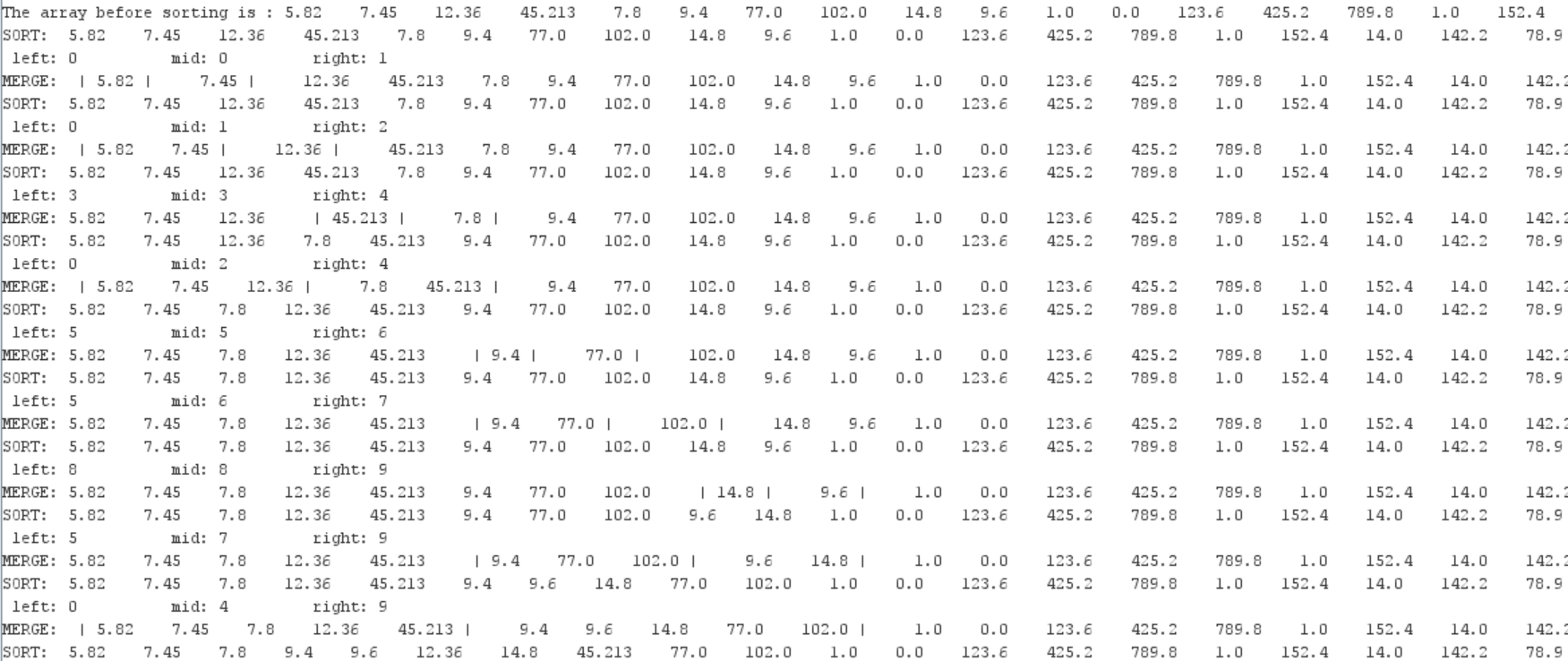


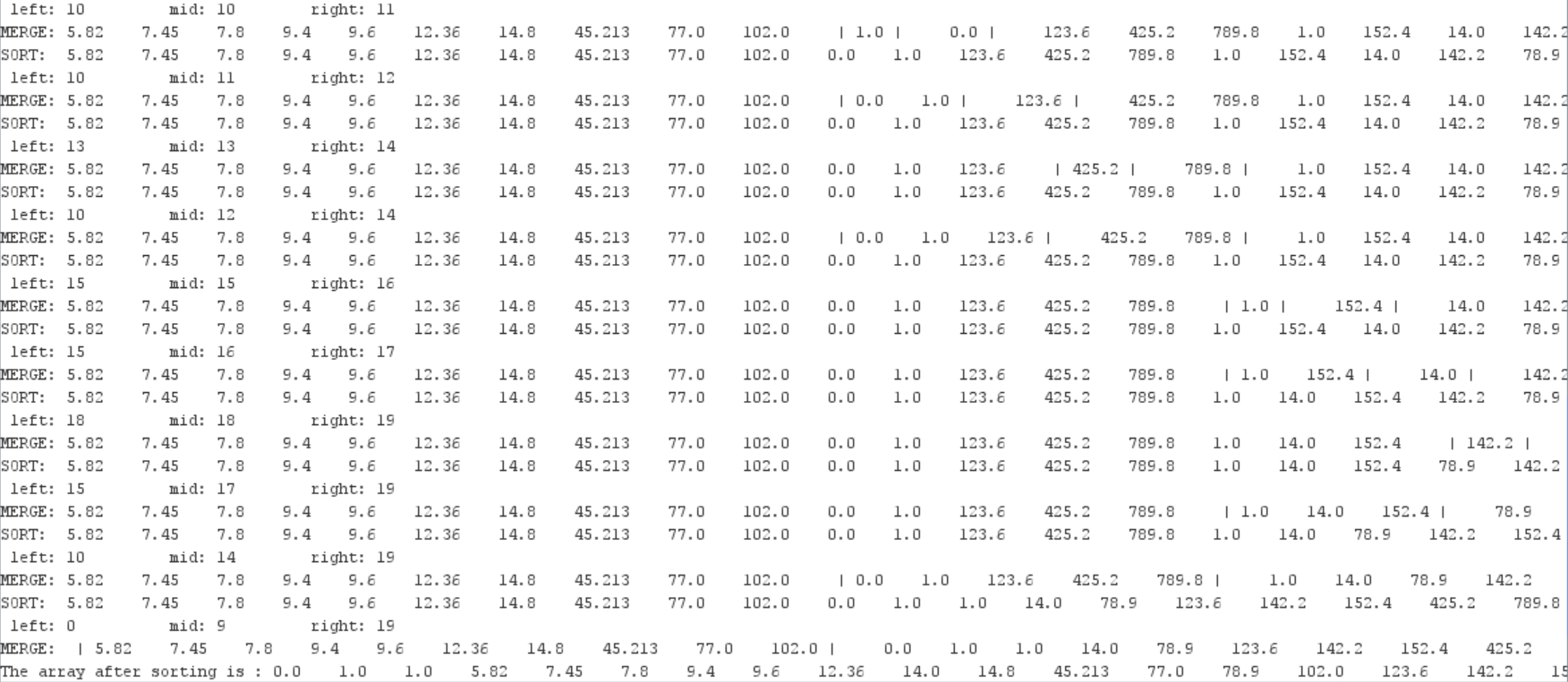
Thời gian thực thi chương trình:

CPU time=IC\*CPI\*clock cycle time=(IC\*CPI)/Clock rate

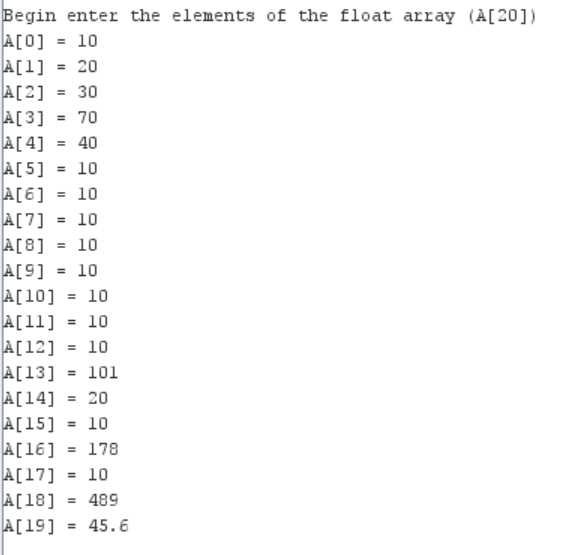
* CPU time= (16569\*1)/2\*10^9=8.28\*10^-6(s)

**Output 7**:

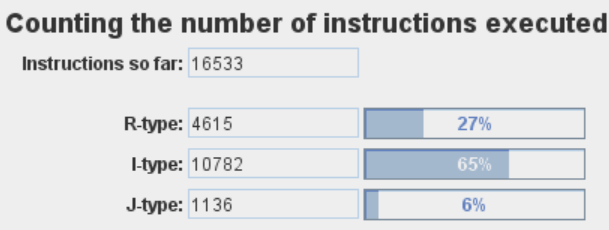




**Input 8:**



Số lệnh thực thi: 16533

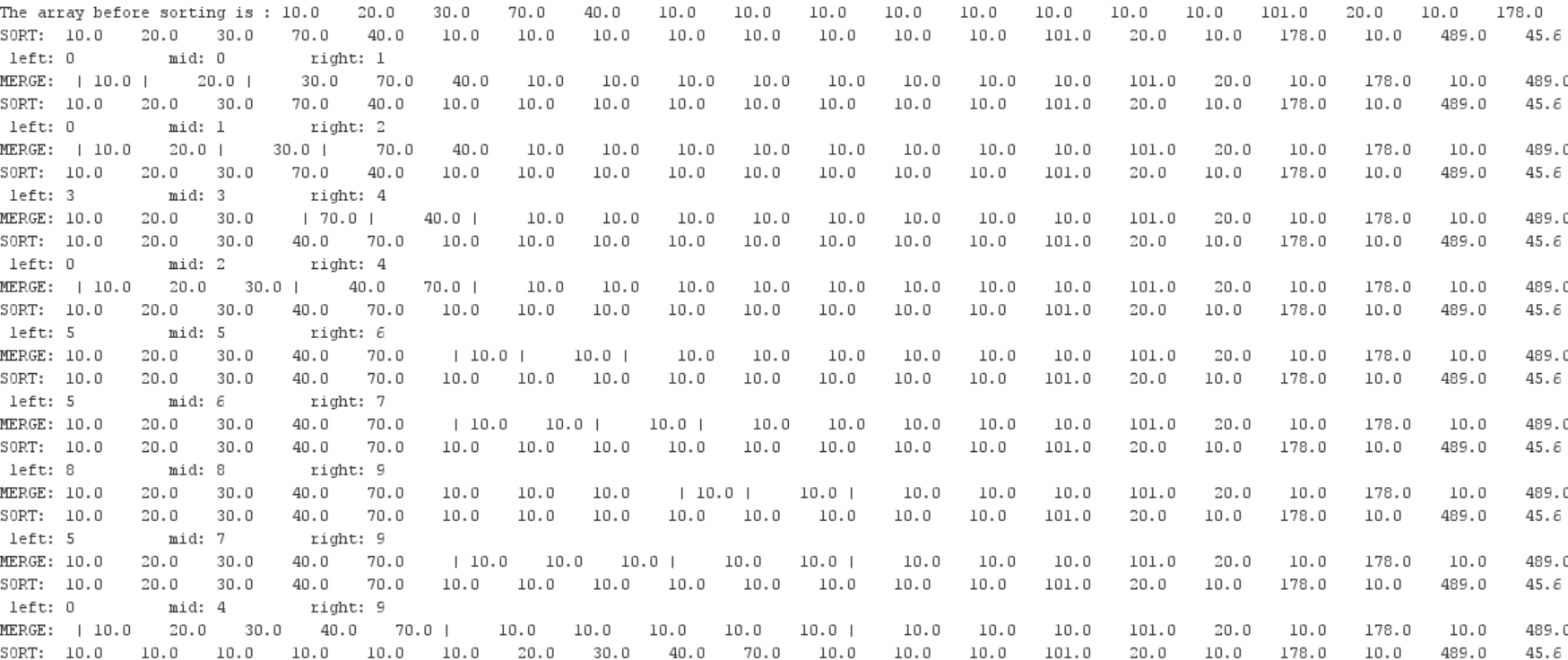


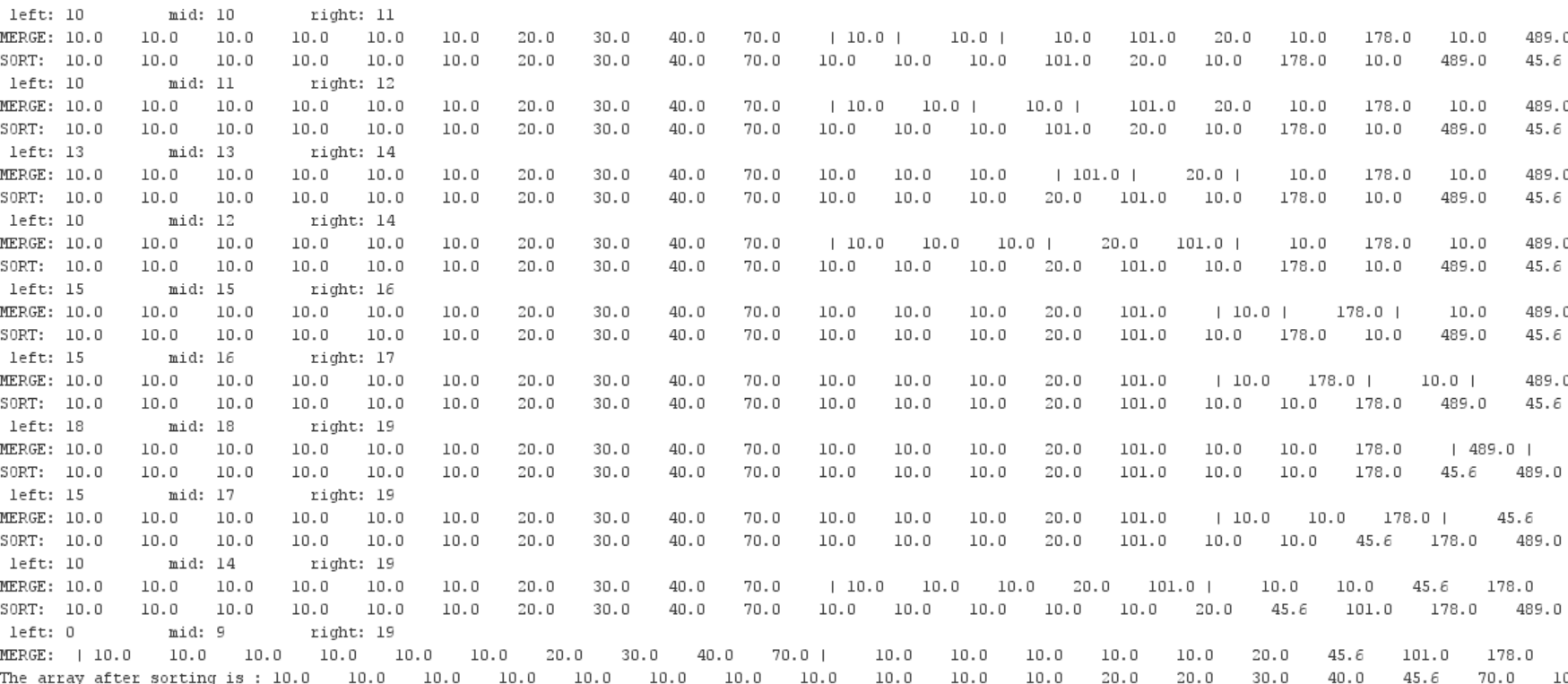
Thời gian thực thi chương trình:

CPU time=IC\*CPI\*clock cycle time=(IC\*CPI)/Clock rate

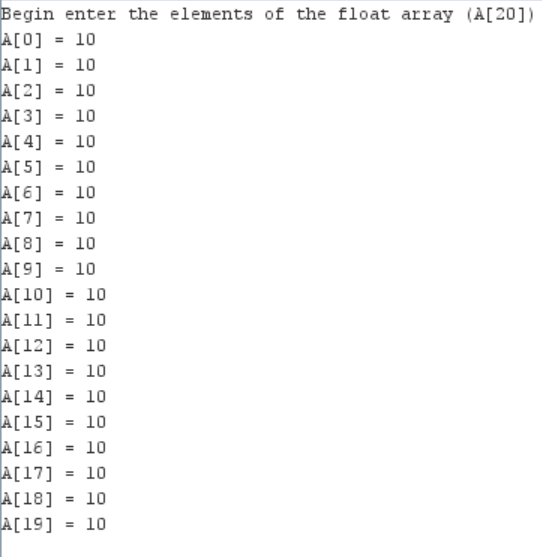
* CPU time= (16533\*1)/2\*10^9=8.27\*10^-6(s)

**Output 8**:

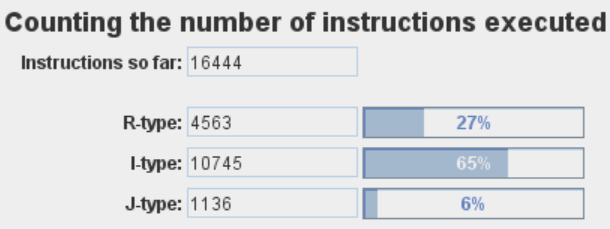




**Input 9:**



Số lệnh thực thi: 16444

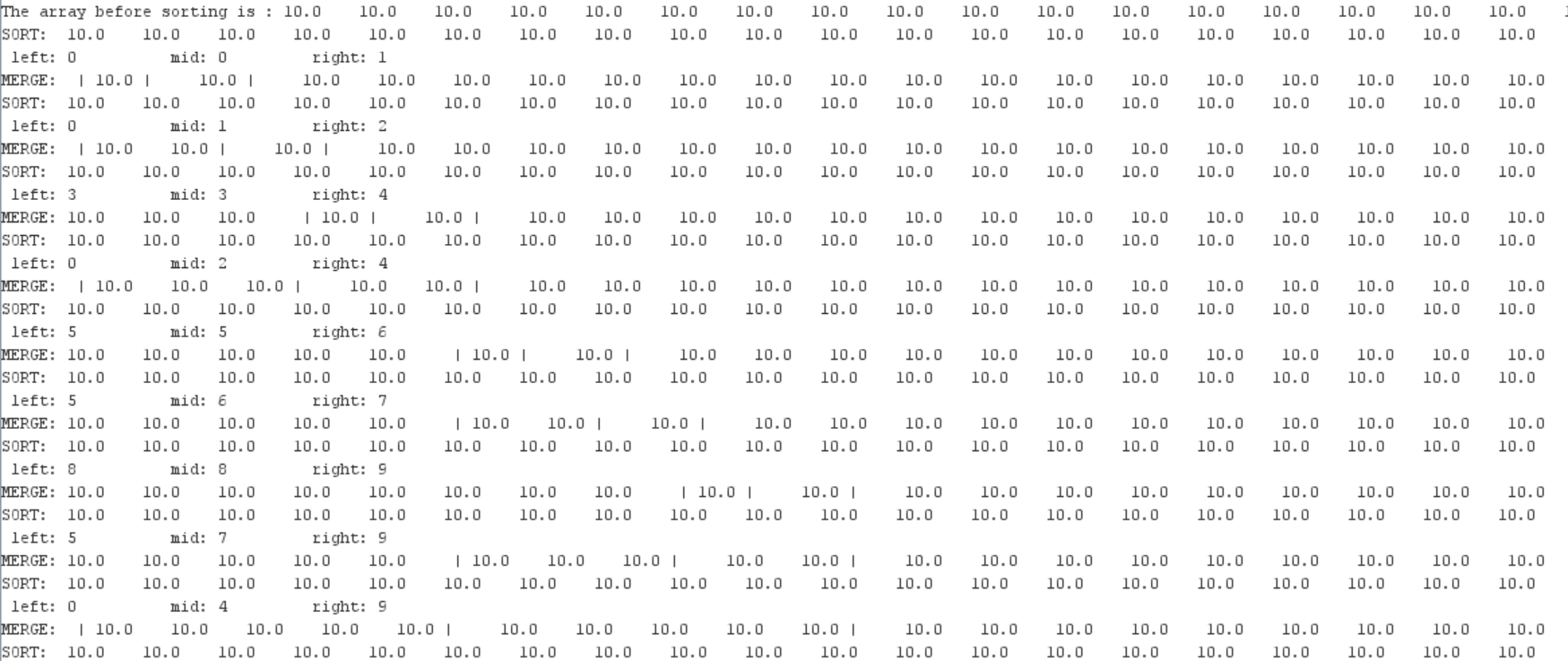


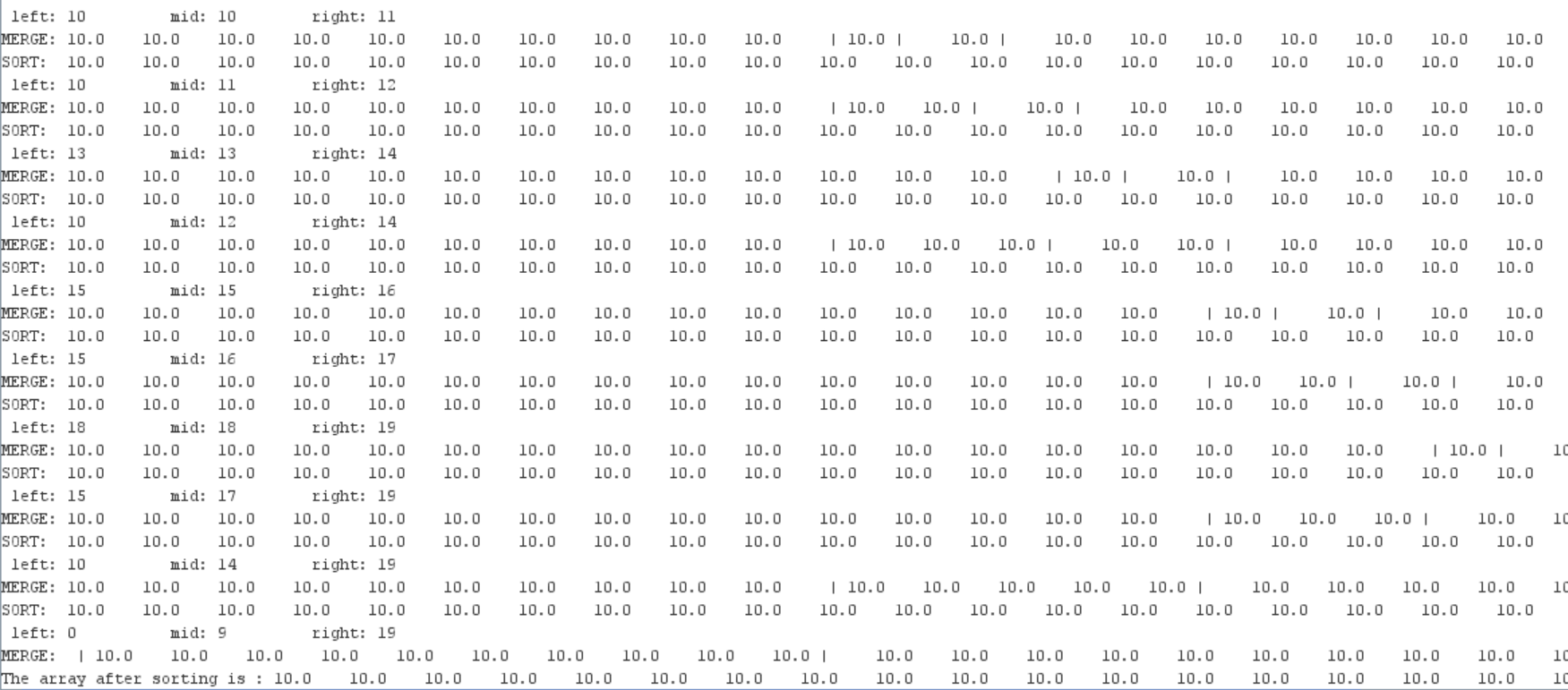
Thời gian thực thi chương trình:

CPU time=IC\*CPI\*clock cycle time=(IC\*CPI)/Clock rate

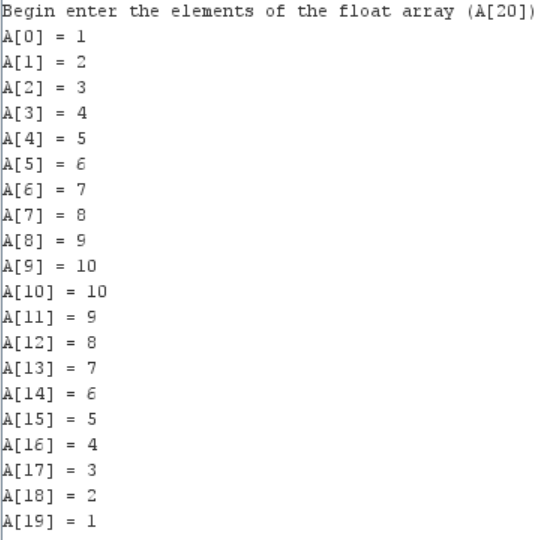
* CPU time= (16444\*1)/2\*10^9=8.22\*10^-6(s)

**Output 9:**

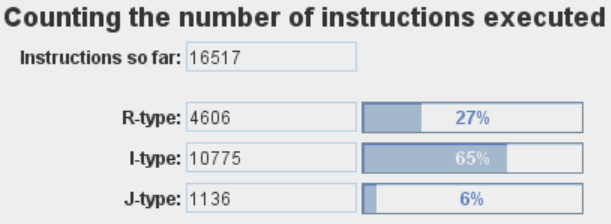
****

****

**Input 10:**



Số lệnh thực thi: 164517

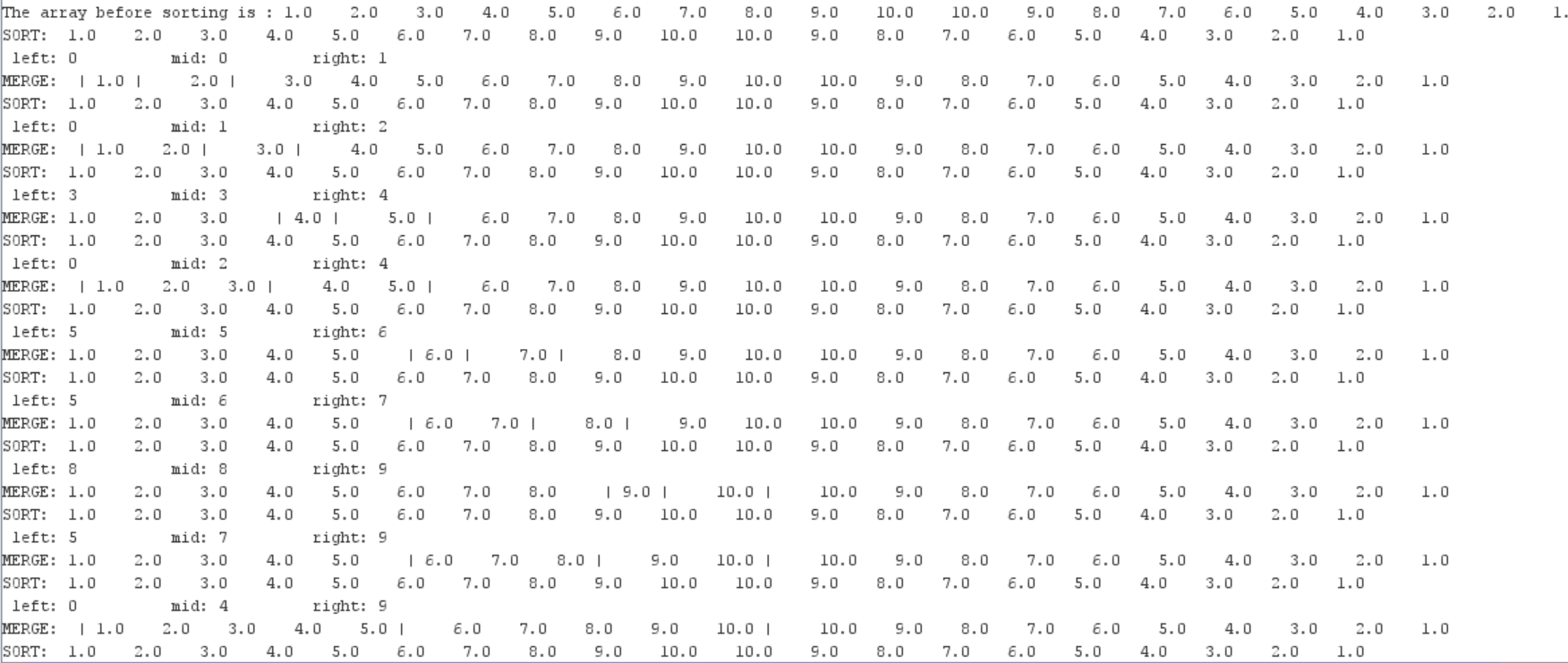


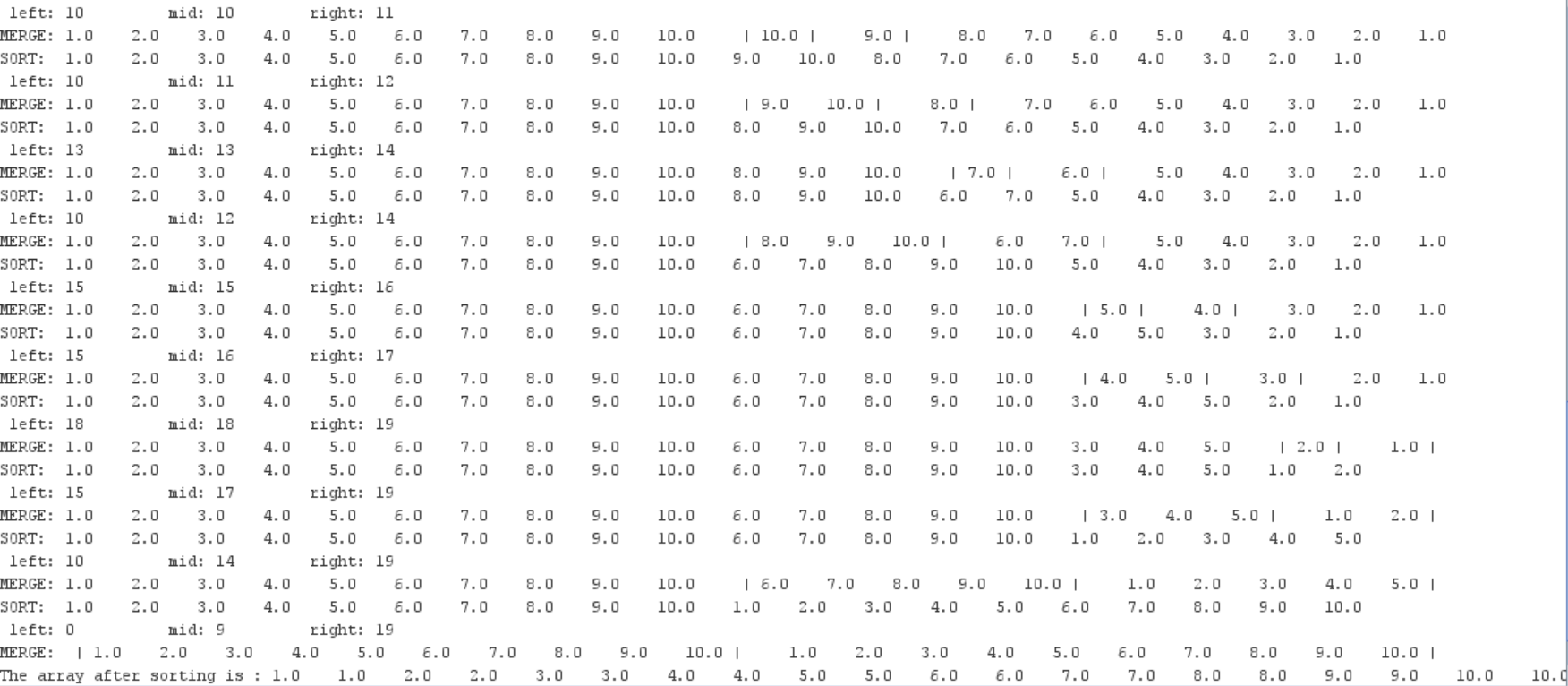
Thời gian thực thi chương trình:

CPU time=IC\*CPI\*clock cycle time=(IC\*CPI)/Clock rate

* CPU time= (16517\*1)/2\*10^9=8.26\*10^-6(s)

**Output 10**:





**Tổng kết**:

|  |  |  |
| --- | --- | --- |
| Input | IC | CPU time (10^-6 s) |
| 1 | 16598 | 8.3 |
| 2 | 16561 | 8.28 |
| 3 | 16462 | 8.23 |
| 4 | 16741 | 8.37 |
| 5 | 16571 | 8.29 |
| 6 | 16599 | 8.3 |
| 7 | 16569 | 8.28 |
| 8 | 16533 | 8.27 |
| 9 | 16444 | 8.22 |
| 10 | 16517 | 8.26 |
|  | Avg=19560 | Avg=8.28 |