

**Ingegneria degli Algoritmi (A.A. 2010-2011)**

Corsi di Laurea in Ingegneria Informatica e Automatica, Ingegneria dei Sistemi Informatici, e Laurea Magistrale in Ingegneria Informatica

*Sapienza Università di Roma*

**Primo appello (15/6/2011) – Durata 2 ore – 5/6 cfu – Compito A**

<p>Cognome: _____</p> <p>Nome: _____</p> <p>Matricola: _____</p>	<p>Autorizzo la pubblicazione del voto di questo esame sul sito web <a href="http://www.dis.uniroma1.it/~demetres/didattica/ae">http://www.dis.uniroma1.it/~demetres/didattica/ae</a>, secondo quanto prevede il decreto legislativo 196/2003 (codice in materia di protezione dei dati personali) che dichiaro di conoscere. In fede,</p> <p>_____</p>
--	---

Rispondere alle seguenti domande, **motivando le risposte** (risposte non motivate saranno considerate nulle).

---

**Domanda 1 [8 punti]**

Si consideri il seguente report di profiling dell'esecuzione di un programma che invoca le tre funzioni *main*, *f* e *g*:

- main*: 10%
- f*: 60%
- g*: 30%

Dopo aver applicato un certo numero di ottimizzazioni, riusciamo a ottenere uno speedup 2x sul tempo di esecuzione della funzione *f*. Rimpiazzando *f* con la sua versione ottimizzata:

1. che speedup otterremmo per l'esecuzione del programma?
2. che report di profiling ci aspetteremmo?

---

**Domanda 2 [10 punti]**

Si consideri un allocatore di memoria di tipo *segregated free lists* con *strict size classes with rounding* che:

- non faccia *splitting/coalescing*
- lavori su una piattaforma a 64 bit
- preveda 29 size di payload distinte: 8, 16, 32, ...,  $2^{31}$
- allinei l'indirizzo iniziale del payload di ogni blocco a un multiplo di 8 byte
- abbia per ogni blocco una header di 4 byte contenente la size del blocco stesso

Illustrare la struttura dell'heap passo-passo durante la seguente sequenza di operazioni, a partire da un heap vuoto:

1. inizializzazione allocatore
2. `b1=malloc(10)`
3. `b2=malloc(15)`
4. `b3=malloc(6)`
5. `b4=malloc(24)`
6. `b5=malloc(5)`
7. `free(b3)`
8. `free(b4)`
9. `b6=malloc(13)`
10. `free(b5)`
11. `b7=malloc(12)`

Includere nei disegni la lookup table e i blocchi comprensivi di header, puntatori, padding, ecc. evidenziando blocchi liberi, blocchi in uso e liste di blocchi liberi.

Si consideri lo stato dell'allocatore alla fine della sequenza e si risponda alle seguenti domande:

- Si ha frammentazione esterna, interna, entrambe, o nessuna?
- Di quanti byte in avanti risulta spostato il `brk` rispetto alla base dell'heap?
- Qual è il fattore di utilizzo dell'allocatore (rapporto tra payload totale e memoria richiesta al sistema operativo mediante `sbrk`)?

---

### Domanda 3 [8 punti]

Si consideri il seguente frammento di codice assembly<sup>1</sup> in sintassi AT&T generato dal compilatore `gcc` su una piattaforma `x86-64` conforme alla System V AMD64 ABI:

```
f:
    pushq %rbp
    movq %rsp, %rbp
    movl %edi, -4(%rbp)
    movl %esi, -8(%rbp)
    movl -8(%rbp), %edx
    movl -4(%rbp), %eax
    subl %edx, %eax
    incl %eax
    leave
    ret
```

Rispondere alle seguenti domande.

1. Con che livello di ottimizzazione è stato compilato il frammento sopra?
2. Scrivere una funzione `f` in linguaggio C la cui compilazione con `gcc -S` fornisca il codice assembly sopra riportato.

---

<sup>1</sup> Consultare la tabella in calce al compito per una descrizione delle istruzioni `x86-64` utilizzate.

---

**Domanda 4 [6 punti]**

Sia `a` una variabile di tipo `int` e sia `b` una variabile di tipo `double*`. Per ciascuna delle seguenti espressioni C dire: 1) se sono valide; 2) che tipo hanno; 3) se denotano lvalue o rvalue.

1. `a+((int* (*)(double*))b)(b+4)`
2. `a+&(b+a)`
3. `a+*&b[2]-9.71)`
4. `*(a+&*b)`
5. `*(int**)&a`
6. `a+*((struct { int* c, d; }*)b)->c)`

prefix	description	example	C analog
add	add source to destination	addl \$5,%ecx	ecx += 5
call	procedure call	call _foo	foo()
cltq	sign-extend <b>eax</b> to <b>rax</b>	–	–
dec	decrement destination	decq %rcx	rcx--
imul	multiply destination with source	imull %esi,%eax	eax *= esi
inc	increment destination	incl %ecx	ecx++
ja	jump if above <i>(unsigned comparison)</i>	cmpl %eax,%ebx ja L2	if ((unsigned)eax > (unsigned)ebx) goto L2
jae	jump if above or equal <i>(unsigned comparison)</i>	cmpl %eax,%ebx jae L2	if ((unsigned)eax >= (unsigned)ebx) goto L2
jb	jump if below <i>(unsigned comparison)</i>	cmpl %eax,%ebx jb L2	if ((unsigned)eax < (unsigned)ebx) goto L2
jbe	jump if below or equal <i>(unsigned comparison)</i>	cmpl %eax,%ebx jbe L2	if ((unsigned)eax <= (unsigned)ebx) goto L2
je	jump if equal	cmpq %rax,%rbx je L2	if (rax == rbx) goto L2
jg	jump if greater <i>(signed comparison)</i>	cmpq %rax,%rbx jg L2	if (rax > rbx) goto L2
jge	jump if greater or equal <i>(signed comparison)</i>	cmpq %rax,%rbx jge L2	if (rax >= rbx) goto L2
jl	jump if less <i>(signed comparison)</i>	cmpq %rax,%rbx jl L2	if (rax < rbx) goto L2
jle	jump if less or equal <i>(signed comparison)</i>	cmpq %rax,%rbx jle L2	if (rax <= rbx) goto L2
jmp	unconditional jump	jmp L2	goto L2
jne	jump if not equal	cmpq %rax,%rbx jne L2	if (rax != rbx) goto L2
jnz	identical to <b>jne</b>	–	–
jz	identical to <b>je</b>	–	–
lea	copy address to destination (load effective address)	leaq -12(%rax),%rcx	rcx=rax-12
leave	pop the current stack frame, and restore the caller's frame	–	–
mov	copy data from source to destination	movq \$7,(%rax)	*(long*)rax=7
movabs	copy 64-bit immediate to destination register	movabsq \$-7,%rax	rax=-7
movsl	copy sign-extended word to destination register	movslq %bx,%rax	rax=bx
movzlw	copy zero-extended word to destination register	movzlw %bx,%eax	eax=(unsigned)bx
movsbq	copy sign-extended byte to destination register	movsbq %bl,%rax	rax=bl
movzbq	copy zero-extended byte to destination register	movzbq %bl,%rax	rax=(unsigned)bl
pop	pop value from stack and write it to destination	popq %rbx	–
push	push value on stack	pushq %rbx	–
ret	return from procedure call	ret	return
sub	subtract source from destination	subl \$5,%ecx	ecx -= 5

Tabella 1: Istruzioni x86-64 più comunemente utilizzate.